Nico Hartto

HTML5-pohjaiset mobiilisovellukset

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

7.5.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Nico Hartto HTML5-pohjaiset mobiilisovellukset 33 sivua 7.5.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Senior Developer Ville Suvilaakso Yliopettaja Kari Salo

Insinöörityössä tutkittiin asiakasyrityksen toimeksiantona, miten HTML5-, CSS3- ja JavaScript-tekniikoita hyödyntävät Apache Cordova- ja Adobe PhoneGap -sovelluskehykset soveltuvat mobiilisovellusten kehittämiseen. Tarkoituksena oli selvittää, mitä vahvuuksia ja heikkouksia näihin sovelluskehyksiin liittyy. Samalla insinöörityön tarkoitus oli tutkia sitä, miten HTML5-, CSS3- ja JavaScript-tekniikoilla toteutetaan yksinkertainen sovellus, joka hyödyntää Wikitude-liitännäistä lisätyn todellisuuden toiminnollisuuden toteuttamiseen. Insinöörityössä tutkittiin, mitä tämä kehitysprosessi pitää sisällään. Vertailua tehtiin myös sen suhteen, miten hybridisovellus Apache Cordova- ja Adobe PhoneGap -sovelluskehyksen päälle rakennettuna vertautuu natiivisovellukseen.

Mobiilisovellusten kehittäminen on usein hankalaa ja hidasta. Osaavien kehittäjien löytäminen ei ole helppoa. On helpompaa löytää kehittäjiä, joilla on jo taustaa HTML5-, CSS3- ja JavaScript-tekniikoista. Useimmiten projektien luonne ei välttämättä ole sellainen, että natiivisovellus olisi kaikkein järkevin lähestymistapa.

Insinöörityön tuloksena selvisi tarkemmin myös se, miten natiivisovellus eroaa hybridisovelluksesta esimerkiksi suorituskyvyn suhteen. Suorituskykyerot ovat merkittäviä, mutta näitä eroja voidaan kaventaa, kun huomioidaan WebView-komponentin rajoitteet piirto- ja reflow-tapahtumien suhteen. Insinöörityön tuloksena saatiin myös tietoa siitä, mitä asioita pitää ottaa huomioon, kun suunnitellaan hybridisovellusta Apache Cordova- tai Adobe PhoneGap -sovelluskehysten päälle, mitä kirjastoja voidaan käyttää hybridisovelluksen kanssa ja mitä kirjastoja on saatavilla vain natiivisovelluksiin.

Insinöörityön lopputuloksena saatu tutkimustieto hyödynnetään jatkossa asiakasyrityksen toiminnan kehittämisessä, ja se tulee vaikuttamaan siihen, mitä tekniikoita valitaan erilaisiin projekteihin. Insinöörityön tutkimuksen kohteena olleet tekniikat ovat yleiskäyttöisiä myös selainsovellusten kannalta, joten tämän insinöörityön tuloksena saatua tutkimustietoa näistä eri mahdollisuuksista hyödynnetään myös tulevissa selainsovellusprojekteissa.

Avainsanat	lisätty todellisuus, mobiilisovellukset, HTML5, Apache Cordova, Adobe PhoneGap, JavaScript

Author Title Number of Pages Date	Nico Hartto HTML5-based mobileapplications 33 pages 7 May 2014
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Ville Suvilaakso, Senior Developer Kari Salo, Principal Lecturer

The thesis studied how to use HTML5-, CSS3-, and JavaScript -technologies with Apache Cordova and Adobe PhoneGap application frameworks, which are suitable for mobile application development. The aim was to find out the strengths and weaknesses of these application frameworks. The thesis explores how HTML5-, CSS3- and JavaScript -technologies are used to implement a simple application that uses Wikitude augmented reality plugin. The thesis describes the development process and methods of said prototype mobile application. Comparisons were also made in terms of how hybrid application built on top of Apache Cordova or Adobe PhoneGap frameworks compares to native application.

Mobile Application development is often difficult and slow. Finding skilled developers to develop native applications is not easy. It is easier to find developers who already have a background in HTML5-, CSS3- and JavaScript -technologies. In many cases, the nature of the project may not be such that a native application would be the most sensible approach.

The thesis also describes in more detail how native application and hybrid applications are differ, for example, in terms of performance. Performance differences are significant, but they can be minimized, given the constraints of the WebView-component in the paint and reflow events. As a result of this thesis, information was obtained about the things to consider when designing a hybrid application built with Apache Cordova or Adobe PhoneGap frameworks.

The information obtained from the research will be utilized in the client company, and the results influence what techniques are selected for various projects. Techniques described in this thesis are widely applicable to different fields beyond mobile application development since the technologies used are also used in web application development.

Keywords	augmented reality, mobile applications, HTML5, Apache Cordova, Adobe PhoneGap, JavaScript

Sisällys

Lyhenteet

AR	Augmented Reality. Lisätty todellisuus.
API	Application Programming Interface, ohjelmointirajapinta.
DOM	Document Object Model, dokumenttioliomalli.
VR	Virtual Reality. Virtuaalinen todellisuus.
FPS	Frames per second. Ruudunpäivitysnopeus.
HTML5	Hyper Text Markup Language 5. HTML-merkintäkielen uusin versio.
WHATWG	Web Hypertext Application Technology Working Group.
JS	JavaScript. Skriptauskieli.
3D	Three-dimensional space. Kolmiulotteinen tila.
POI	Point of interest. GPS-koordinaattien avulla toteutettu lisätty todellisuus.
GPS	Global Positioning System. Maailmanlaajuinen paikallistamisjärjestelmä.
GNSS	Galileo. Euroopan unionin (EU) ja Euroopan avaruusjärjestön (ESA) ylläpitämä paikannusjärjestelmä.
GLONASS	Globalnaja navigatsionnaja sputnikovaja sistema. Neuvostoliiton/Venäjän ylläpitämä paikannusjärjestelmä.
IR	Image Recognition. Kuvantunnistus.
IT	Image Tracking. Kuvanseuraus.
SDK	Software development kit. Sovelluskehityskirjasto.

B2C Business-to-customer. Kuluttajille suunnattu kulutushyödykkeiden markkinointi.

B2B Business-to-business. Yritysten toiminnassa tarvittavien tuotantohyödykkeiden markkinointi.

1 Johdanto

Insinöörityön tarkoituksena on ohjelmoida ja tuottaa toimeksiantajalle Apache Cordova- tai Adobe PhoneGap -sovelluskehystä käyttävä prototyyppisovellus ja kerätä tutkimustietoa erilaisista lisätyn todellisuuden ratkaisuista mobiilisovelluksissa sekä itse sovelluskehyksistä. Tämän sovelluksen tekemisestä saatua tutkimustietoa hyödynnetään tulevissa projekteissa ja niiden mielekkyyden arvioinnissa.

Teknisten ratkaisujen taustalla on usein kolmannen osapuolen kehittämiä kirjastoja, sovelluskehyksiä, SDK:ita eli sovelluskehityskirjastoja, ja sujuvan kehitysprosessin kannalta on ensiarvoisen tärkeää valita tilanteeseen parhaiten soveltuvat työkalut siitä suuresta määrästä, mitä on tarjolla. Molemmat, Apache Cordova ja Adobe PhoneGap, ovat olleet pitkään erittäin suosittuja hybridisovelluskehittäjien parissa. Apache Cordovan ja Adobe PhoneGapin avulla pystytään kirjoittamaan sovellus HTML5-, JavaScript- ja CSS3-tekniikoita käyttäen sekä hyödyntämään mobiilikäyttöjärjestelmän laitetasonominaisuuksia JavaScript-ohjelmointirajapinnan avulla.

Kuvantunnistus on yksi sovelluskehityksen alue, jossa juuri nyt tapahtuu paljon ja erittäin nopeasti. Nykyään erilaisia sovelluskirjastoja ja sovelluskehityskirjastoja on saatavilla melkein kaikille alustoille. Alalla on useita tarjoajia, ja läheskään aina työkalut eivät ole ilmaisia. Kattavan vertailun tekeminen on aikaa vievää ja vaatii tiettyä tietotaitoa, jotta tuloksesta on hyötyä.

Insinöörityön toimeksiantaja on Kuubi Visual Productions Oy, joka on yksi Suomen vanhimmista digitaaliseen mediaan erikoistuneista yrityksistä. Sen historia juontaa Riot Entertainment Oy:n alkuaikoihin, ja monet nykyisistä omistajista ovat lähtöisin Riot Entertainmentin markkinointipuolelta. Nykyään yrityksessä työskentelee 20 media-alan ammattilaista. Kuubi Visual Productions Oy:n pääasiallinen toimiala on tuottaa asiakkaille erilaisia digitaalisen markkinoinnin ratkaisuja.

Alkuvaiheessa kävimme Kuubi Visual Productions Oy:n edustajien kanssa läpi suuripiirteisesti sen, mitä sovelluksen tulisi tehdä. Yrityksessä on aiemmin tehty Adobe PhoneGap -projekteja, mutta ei sellaista jossa olisi tuotu mukaan lisätyn todellisuuden elementtejä. Prototyyppisovelluksen lisätyn todellisuuden sovelluskehyksenä on Wikitude-

alusta. Se puolestaan hyödyntää Qualcomm Vuforia -sovelluskehysalustaa lisätyn todellisuuden tarvitsemaa kuvantunnistusta varten.

Kuubi Visual Productions Oy:n tekeminen ei kuitenkaan rajoitu pelkästään digitaaliseen markkinointiin, vaan Kuubi toteuttaa monenlaisia digitaalisia sisältöjä aina asiakkaan tarpeiden mukaan. Projektien koko vaihtelee muutaman päivän projekteista aina pidempiin useamman kuukauden mittaisiin projekteihin. Kuubi Visual Productions Oy toteuttaa niin mobiilisovelluksia kuin verkkosivustoja ja laajempia ohjelmistokehitysprojekteja asiakkaille. Yleensä painopiste on kuitenkin kokeilevalla puolella ja uusien teknologioiden parissa. Näiden teknologioiden avulla tuodaan asiakkaan liiketoimintaan lisäarvoa eri muodoissa, oli tämä lisäarvo sitten brändin tunnettavuuden lisääminen, suora B2C-palvelun kehittäminen tai B2B-puolelle tuotu ratkaisu yritysten väliseen toimintaan.

Mobiilisovellukset ovat nykyään isossa roolissa kaikkialla, ja alati kasvavat markkinat tuovat koko ajan uusia käyttäjiä eri mobiilialustoille. Näille markkinoille murtautumista varten tarvitaan kuitenkin kustannustehokas tapa lähteä tekemään sovelluksia. Asiakkaiden tarpeisiin pystytään useimmiten vastaamaan tekemällä sovellus niin sanottuna hybridisovelluksena. Näissä sovelluksissa kirjoitetaan lähdekoodi vain kerran, ja lähdekoodi käännetään eri alustojen ymmärtämään muotoon ohjelmistokehyksen avulla.

Nykyään markkinoilla on monia eri mobiilikäyttöjärjestelmiä, joista kolme suurinta ovat Microsoft Windows Phone, Google Android ja Apple iOS. Suomalainen Jolla on myös tuonut oman Sailfish-mobiilikäyttöjärjestelmänsä markkinoille. Se pohjautuu Nokian kehittämään Meego-käyttöjärjestelmään. Koska mobiilikäyttöjärjestelmiä on niin paljon, pienempien mobiilisovellusten tekeminen usealle alustalle on usein hyvin haastavaa vaaditun ohjelmointi- ja alustaosaamisen takia. Kustannustehokkaampaa on tehdä sovellus kerran, ja välissä oleva Apache Cordova- tai Adobe PhoneGap-sovelluskehys hoitaa ohjelman kääntämisen tuettujen alustojen ymmärtämään muotoon.

Myös prototyyppien rakentaminen on taloudellisesti kannattavampaa, sillä koska Apache Cordova- ja Adobe PhoneGap -sovellukset pohjautuvat HTML5-, CSS3- ja JavaScript-tekniikoihin, useilla kehittäjillä jo taustaa näissä tekniikoissa ja tämän osaamisen siirtäminen mobiilipuolelle on huomattava etu. Yhdellä lähdekielellä tehdyn lähdekoodin hyödyntäminen usealla alustalla on selkeä kustannussäästö yritykselle.

Insinöörityön tavoitteena on tuottaa prototyyppisovellus, jonka pohjalta pystytään arvioimaan, onko Wikitude-liitännäinen paras ratkaisu lisätyn todellisuuden sovelluksiin. Insinöörityön tarkoituksena on myös samalla vertailla erilaisten lisätyn todellisuuden sovelluskehysten soveltuvuutta pienen ohjelmistotalon tuotantoihin sekä tutkia HTML5-, CSS3-, JavaScript-tekniikoiden ja Apache Cordova-, Adobe PhoneGap-sovelluskehysten sekä Wikitude-liitännäisen soveltuvuutta toiminallisuudeltaan yksinkertaisempien mobiilisovellusten kehitykseen.

Prototyyppisovelluksen perusajatus on tuoda Kuubi Visual Productions Oy:n käyntikorttien yhteyteen lisätyn todellisuuden sisältöä; käyntikortti toimii tässä tapauksessa siis kuvatuntisteena. Protyyppisovelluksen tarkoituksena on tuoda esiin lisätyn todellisuuden avulla yksinkertaista HTML-pohjaista verkkosisältöä sovelluksen sisällä.

Wikitude-liitännäinen valittiin tekniseksi ratkaisuksi siksi, että se on saatavilla Apache Cordova- ja Adobe PhoneGap -alustalle. Wikitude pohjautuu Qualcomm Vuforiaan, ja Qualcomm Vuforia on myös jo pitkään kehityksessä ollut ilmainen lisätyn todellisuuden ympäristö, josta dokumentaatiota on saatavilla riittävästi. Qualcomm Vuforiaa ei ole suoraan saatavilla Apache Cordova- tai Adobe PhoneGap -sovelluskehyksille, joten tämän takia oli löydettävä erillinen ratkaisu, jossa tällä kertaa taustalla teknisenä ratkaisuna kuitenkin on Qualcomm Vuforia.

2 Lisätty todellisuus

Lisätty todellisuus (AR, augmented reality) tarkoittaa sitä, että olemassa olevaan maailmaan tuodaan virtuaalista lisäsisältöä. Teknisesti tämä voidaan saavuttaa eri tavoilla, mutta yleisin tapa on tuoda kameran kuvaaman oikean maailman videokuvan päälle 3D-malleja tai muuta sisältöä. Toinen tapa on projisoida fyysiselle pinnalle 3D-sisältöä. Tämä tekniikka tosin ei ole kuluttajille saatavilla rajoittuneisuutensa takia. Projisointi tarvitsee suuren tilan, joka toimii pintana, johon heijastetaan virtuaalinen sisältö. Cave-tyyppiset ratkaisut toimivat myös virtuaalisen todellisuuden (VR, virtual reality) pohjana. Nämä myös perustuvat projisointitekniikkaan. Nykyään kuluttajien saatavilla on myös kolmas vaihtoehto, silmien eteen asetettavat lasit. (Azuma 1997.)

Google on tullut lisätyn todellisuuden markkinoille omalla tuotteellaan, Google Glass. Google Glass sisältää sangat, joissa toisen silmän eteen tulee pieni linssi, johon kaikki

lisätyn todellisuuden sisältö tulee näkyviin. Oikealla puolella kehyksiä on myös pieni kosketuspinta, jonka kautta käyttäjä voi ohjata Google Glass -sovelluksia. Google Glass pohjautuu Googlen omaan Android-käyttöjärjestelmään. (Google Glass 2014.)

Virtuaalisen todellisuuden alalla on ollut historian aikana useita yrittäjiä, mutta kaupallista menestystä ei saavutettu teknisesti rajoittuneilla laitteilla. Nykyään on sovelluskehittäjien ja harrastelijoiden saatavilla kuitenkin teknisesti riittävän pitkälle kehittynyt virtuaalisen todellisuuden ratkaisu, Oculus VR -yrityksen kehittämä Oculus Rift. Nämä virtuaalisen todellisuuden lasit ovat olleet varsin suosittuja. Oculus Rift sai alkunsa kickstarter.com:n kautta yleisörahoitteisena projektina. Nykyään yrityksen teknisenä johtajana toimii John Carmack, joka tunnetaan paremmin yhtenä Wolfenstein 3D- ja Doom-pelien ohjelmoijista ja id Softwaren perustajana. Facebook osti Oculus VR:n koko osakekannan maaliskuussa 2014. (John Carmack joins Oculus VR 2013; Zuckenberg 2014.)

2.1 Kuvantunnistus

Suurin osa lisätyn todellisuuden ratkaisuista pohjautuu kuvantunnistukseen (IR, image recognition). Kuvantunnistus perustuu kuvainformaation analysointiin erilaisten monimutkaisten algoritmien avulla. Kuvantunnistuksen perusperiaatteisiin kuuluvat esimerkiksi seuraavat tekniikat:

- kuvion etsintä, tietyn kuvion etsiminen, mallin sovittaminen kuvaan ja löytöjen määrän laskeminen
- pikselien määrän laskenta, lasketaan tietyltä alueelta tummien ja vaaleiden pikselien määrä
- kynnystys, kuvan sävyjen määrän vähentäminen valkoiseen ja mustaan
- reunan haku, kohteen reunojen etsiminen kuvasta esimerkiksi Sobel filtteröinnin avulla. (Forsyth & Ponce 2003.)

Nykyään jo mobiililaitteissa, kuten esimerkiksi älypuhelimissa, on laskentatehoa riittävästi siihen, että tämä kuvantunnistusalgoritmi voidaan ajaa useita kertoja sekunnissa ruudun päivitysnopeuden mukaan (FPS, Frames per Second). Kuvantunnistus toimii pohjana sille, että lisätyn todellisuuden sisältö pystytään tuomaan oikeaan kohtaan maailmassa. Toisena vaihtoehtona on tuoda sisältöä ilman tunnistetta oikean maailman ”päälle”. Tämä tarkoittaa esimerkiksi sitä, että hyödynnetään mobiililaitteen GPS/GNSS/GLONASS/Wi-fi-pohjaista paikannusta. Tällöin sisältö näkyy koordinaattien

perusteella eikä kuvantunnistusta käytetä tuomaan lisätyn todellisuuden sisältöä näkyviin. (Forsyth & Ponce 2003.)

2.2 Kuvatunniste

Näitä kuvia, jotka laukaisevat lisätyn todellisuuden tunnistimen, kutsutaan usein kuvatunnisteiksi (engl. marker). Kuvatunnisteet voivat olla lähes mitä vain, mutta kuvantunnistukseen liittyvät haasteet rajoittavat kuvantunnistuksen laatua. Ongelmallisia kuvatunnisteita ovat esimerkiksi sellaiset, joissa kontrasti on heikko, pinta on melkein samaa väriä, eivät sisällä teräviä muotoja missään, koostuvat toistuvista kuvioista tai joissa on pyöreitä reuna-alueita. (Image Targets 2014.)

Koska kuvatunniste määrittää lisätyn todellisuuden virtuaalisen sisällön sijainnin, on tärkeää myös seurata kuvaa. Tässä tapauksessa puhutaan kuvanseurauksesta (IT, image tracking). Koska virtuaalisen sisällön ja kameran keskinäinen etäisyys vaihtelee, sovellus skaalaa lisätyn todellisuuden sisältöä aina keskinäisen etäisyyden mukaan. Kuvatunnisteesta myös saadaan tieto siitä, missä kulmassa sisältö on ja mikä akseli (x, y vai y) on ylöspäin. (Image Targets 2014.)

Lisätyn todellisuuden sijoittaminen suhteessa siihen, mitä käyttäjä näkee ruudullaan, on monen monimutkaisen osatekijän summa. Ohjelmistokehityksessä ja tietokoneilla grafiikkaa piirrettäessä on käytössä niin sanottu suorakulmainen koordinaatisto, joka koostuu x-, y-, z-akseleista. Tämäntyyppinen kolmiulotteinen koordinaatisto on useimmille tuttu matematiikan puolelta. Suorakulmaisen koordinaatiston avulla voidaan laskea lisätyn todellisuuden sisällön sijainti suhteessa käyttäjän kameran läpi näkemään maailmaan. Kolmiulotteisen koordinaatiston avulla tätä sisältöä voidaan myös esittää siten, että sen sijainti ei ole suoraan kuvatunnisteen origossa. (Abrash 1997.)

Nykyinen tietokoneiden kolmiulotteisen grafiikan piirto pohjautuu hyvin vahvasti tähän periaatteeseen, ja laitetason ratkaisuja on ehditty vuosikymmeniä kehittää olemaan nopeita tämäntyyppisessä laskennassa. Samat periaatteet pätevät myös mobiiliympäristössä. (Abrash 1997.)

Kolmiulotteisen, pääasiassa vektoreista koostuvan grafiikan muuntamista näytölle kutsutaan rasteroinniksi. Sen vaihtoehtona on ollut jo vuosia säteenseuranta, mutta koska

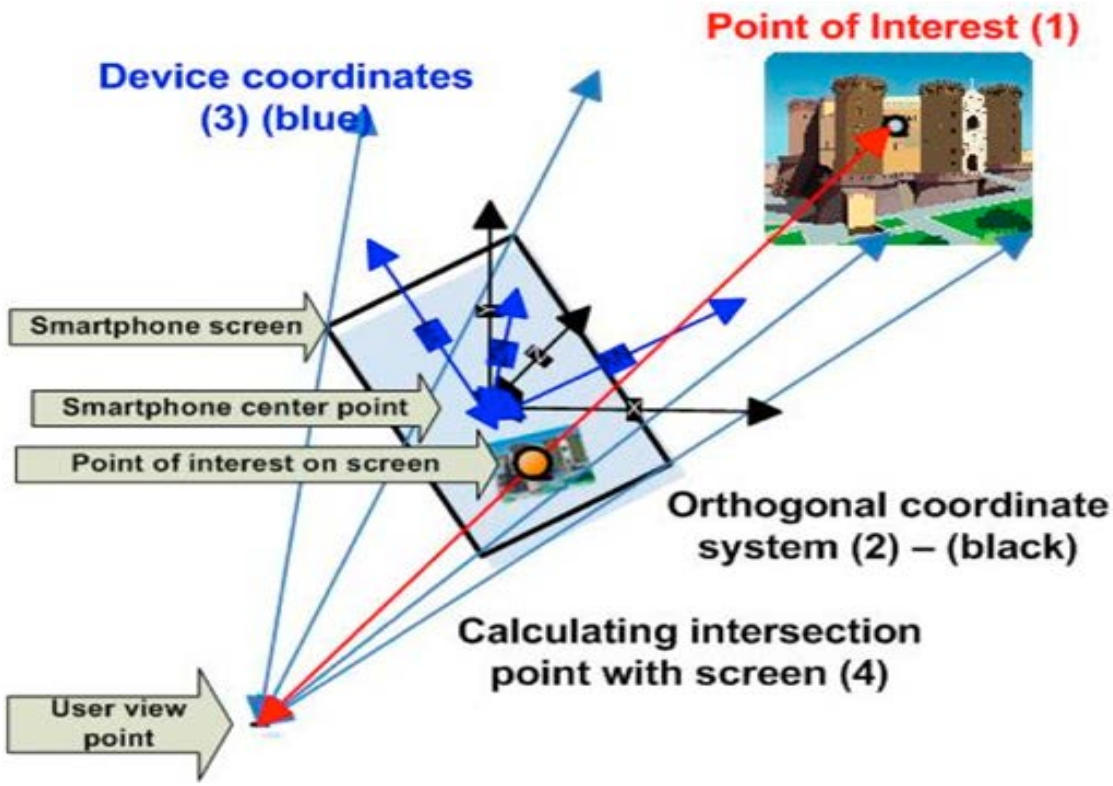
se on suorituskyvyltään huomattavasti heikompi, on rasterointi nykyisinkin suositumpi. Säteenseuranta mahdollistaisi tarkemman mallinnuksen valon kulusta erilaisten materiaalien pinnalla ja niiden välillä. (Abrash 1997.)

Yksi suurimmista kuvantunnistuksen parissa työskentelevistä yrityksistä on Qualcomm Technologies. Se on kehittänyt jo usean vuoden ajan omaa kuvantunnistuksen ja lisätyn todellisuuden ratkaisua nimeltä Vuforia. Vuforia on tällä hetkellä yksi käytetyimmistä pohjaratkaisuista ilmaisuuden ja laajan sovelluskirjastovalikoimansa takia. (Why Vuforia 2014.)

2.3 Wikitude-liitännäinen

Insinöörityössä esiteltävän Ember.js-, Apache Cordova- ja Adobe PhoneGap-, HTML5-, JavaScript-, CSS3-pohjaisen sovelluksen yksi erikoisuus oli Wikitude-liitännäinen ja sen hyödyntäminen lisätyn todellisuuden lisäämiseksi Adobe PhoneGap -pohjaiseen sovellukseen.

Wikitude on Wikitude GmbH -nimisen yrityksen kehittämä lisätyn todellisuuden ratkaisu. Se on saatavilla useille eri alustoille, mukaan lukien Apache Cordova ja Adobe PhoneGap. Wikitude oli yksi ensimmäisistä lisätyn todellisuuden sovelluskehityskirjastoista, joissa oli tuki Wi-fi-, GPS-, Glonass- tai Galileo-paikannuksen perusteella näytetylle lisätylle todellisuudelle. Ensimmäinen versio julkaistiin vuonna 2008, ja vuosina 2008–2012 Wikitude voitti useita palkintoja lisätyn todellisuuden alalla. (Wikitude Awards 2014; Wikitude About 2014.)



Kuva 1. POI-datan näkyminen lisätyn todellisuuden näkymässä (Wikitude GmbH 2011).

Kuvassa 1 näkyy se, miten POI-data (Point of Interest) sijoitetaan lisätyn todellisuuden näkymään. Wikitude käyttää taustalla kuvantunnistuskirjastonaan Qualcomm Vuforiaa. Wikitude GmbH on rakentanut Wikitude-alustaan monia ominaisuuksia, kuten hallintakäyttöliittymän, jolla hallitaan kuvatunnistekirjastoja, jotka ladataan laitteelle. Tämän Wikitude-hallintakäyttöliittymän kautta on myös mahdollista lisätä 3D-malleja ja POI-dataa. (Madden 2012.)

3 Monialustaiset mobiilisovelluskehykset

Suomalaisen Jolla-yhtiön Sailfish-mobiilikäyttöjärjestelmän sovelluksia on mahdollista tehdä Sailfish Silica -sovelluskehyskirjastoa hyödyntäen. Sailfish Silica on laajennus Qt 5.0- ja QtQuick 2 -sovelluskehyskirjastoille. Sailfish-sovellukset voidaan kirjoittaa JavaScriptin ja QML:n eli Qt Modeling Language avulla, joka muistuttaa hyvin pitkälti JavaScriptiä. QML määrittelee sovelluksen ulkoasun, kun taas JavaScriptillä voidaan kirjoittaa sovelluksen logiikka. Sovellusta voidaan myös laajentaa C- tai C++-pohjaisella lähdekoodilla. (Creating applications with Sailfish Silica 2014; Svenn 2014.)

Maailmalla on muitakin näitä samoja tekniikoita käyttäviä projekteja, erikoisempina ehkä Tizen-projekti, jossa koko mobiilikäyttöjärjestelmän sovellukset rakennetaan HTML5-, CSS3- ja JavaScript-tekniikoita hyödyntäen. Tizen-sovelluksia on myös mahdollista tehdä Qt-sovelluskehyksen päälle. Tietysti laitetason sovituskerrokset ja muut korkeaa suorituskykyä vaativat asiat on toteutettu asiaan kuuluvalla tavalla käännetyllä ohjelmointikielellä. Tizen-käyttöjärjestelmää voidaan käyttää useilla eri päätelaitteilla. (Tizen Project 2014; Svenn 2014)

Appcelerator Titanium on yksi maailmalla hyvin tunnetuista sovelluskehyksistä monialustaiseen mobiilisovelluskehitykseen. Se sisältää tuen Android-, BlackBerry-, iOS- ja Tizen-mobiilikäyttöjärjestelmille. Appacelerator tarjoaa myös kattavasti erilaisia pilvipalveluita Titanium-alustan tueksi, kuten esimerkiksi push-notifikaatiot. Appcelerator Titanium on avoimen lähdekoodin alusta. (Titanium Mobile Development Environment 2014.)

Suomalainen AppGyver-niminen yritys työstää myös omaa hybridisovelluskehystään nimeltä Steroids. AppGyver Steroidsin erona esimerkiksi tässä insinöörityössä esiteltyihin Adobe PhoneGap- ja Apache Cordova -sovelluskehyksiin on se, että se myös tarjoaa suoraan kääntöpalvelun, sovelluskomponentit ja liitännäiset. AppGyver Steroids pystyy kutsumaan JavaScript-ohjelmointirajapinnan avulla natiiveja mobiilialustan käyttöliittymäkomponentteja. Tässä insinöörityössä päädyttiin kuitenkin rajaamaan AppGyver Steroids pois maksullisuuden takia. (AppGyver 2014.)

3.1 Hybridisovellukset ja suorituskyky

Hybridisovellus suoritetaan mobiilikäyttöjärjestelmän tarjoaman WebView-komponentin sisällä. WebView on eräänlainen pieni internetselain. Hybridisovellus ei myöskään ole pelkkä HTML5-verkkosovellus, vaan se pystyy käyttämään laajennusten avulla mobiilikäyttöjärjestelmän tarjoamia ohjelmointirajapintoja (API), kuten esimerkiksi kameraa. (LeRoux 2013.)

Adobe PhoneGap -sovellukset eivät käytä mobiilialustan natiivia selainta, vaan WebView-komponenttia. Tästä syystä Adobe PhoneGapin päälle rakennetut sovellukset eivät pääse hyödyntämään paremmin optimoitua JavaScript-tulkkia, joka löytyy natiivista selaimesta. Tämä merkitsee sitä, että Adobe PhoneGap avulla rakennettu sovellus ei

toimi yhtä nopeasti kuin jos sovellus suoritettaisiin puhtaasti natiivissa selaimessa. Natiivin selaimen JavaScript-tulkki on pidemmälle optimoitu ja tämän optimoinnin ansiosta paljon suorituskykyisempi kuin WebView-komponentit käytössä oleva JavaScript-tulkki.

Android-puolella JavaScript-tulkki voi olla Googlen kehittämä V8 JavaScript Engine tai optimoimaton JavaScript-tulkki. Tämä riippuu Android-versiosta ja laitevalmistajasta. Uudemmat Android-puhelimet käyttävät hyödykseen Google V8 JavaScript Engine JavaScript -moottoria. (Chrome V8 2014.)

Suorituskyky on tärkeä asia niin käytettävyyden kannalta kuin kaupallisesta näkökulmasta. Apple ei päästä hitailta tuntuvia sovelluksia Apple AppStoreen. Applen hyväksyntäprosessissa käydään jokainen sovellus läpi, ja jos sovellus ei täytä Applen määrittelemiä ehtoja AppStoreen hyväksynnälle, ei sovellusta julkaista Apple AppStoressa. Googlen omaan sovelluskauppaan Google Play Storeen lisätyn sovelluksen ei tarvitse käydä vastaavaa hyväksyntäprosessia lävitse.

3.2 Apache Cordova- ja Adobe PhoneGap -sovelluskehykset

PhoneGap-projekti sai alkunsa vuonna 2009 iPhoneDevCamp-tapahtumassa San Franciscossa. Adobe osti PhoneGap-projektin ympärille perustetun Nitobi Software -yrityksen vuonna 2011. Samaan aikaan Adobe lahjoitti PhoneGap-projektin lähdekoodin Apache Foundation -järjestölle, ja tämän pohjalta Apache perusti Apache Cordovan.

Monille tämä hieman erikoinen järjestely hämärtää sitä, mitä Apache Cordova tarkoittaa ja mitä Adobe PhoneGap. Pohjimmiltaan kyseessä on yksi ja sama projekti: Adobe PhoneGap perustuu Apache Cordovaan, ja Adobe PhoneGap on lisätty Adoben toimesta tiettyjä ominaisuuksia. Adobe on myös rakentanut työkaluja ja pilvipalveluita PhoneGap-sovelluskehyksen ympärille.

Apache Cordova on avoimen lähdekoodin projekti, ja Adobe on kannustanut kaikkia yrityksiä osallistumaan Adobe PhoneGapin kehitystyöhön Apache Cordova -projektin kautta. Apache Cordova -projektiin tulleet parannukset tulevat myöhemmin aina käyttöön Adobe PhoneGap -sovelluskehyksen kanssa. Tässä insinöörityössä viitataan jatkossa molempiin sovelluskehyksiin Adobe PhoneGap -nimellä. (Apache Cordova 2014.)

Apache Cordova tukee kattavasti eri alustoja, ja Cordova tukee myös monia sellaisia alustoja, joille Adobe PhoneGapista ei löydy tukea. Taulukossa 1 näkyy Apache Cordovan Core- ja Horizon-tasolla tuetut alustat. Core-tasolla olevilla alustoilla on kaikkein laajin tuki Apache Cordovan sisällä. Tämä tarkoittaa sitä, että suurin osa laitetason ominaisuuksista saadaan käyttöön JavaScript-ohjelmointirajapinnan avulla. (Apache Cordova Platform Support.)

Taulukko 1. Apache Cordova -sovelluskehyksen tukemat alustat Core- ja Horizon-tasolla (Apache Cordova Platform Support).

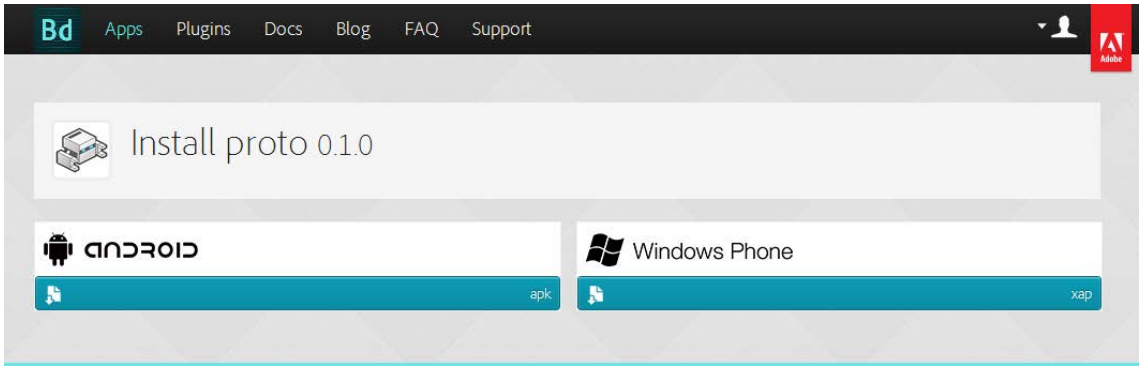
Core	Horizon
iOS	Tizen
Android	Qt
BlackBerry	Firefox OS
Windows Phone	Ubuntu Mobile (Qt)
Windows 8	Windows (Win32)

Adobe PhoneGap -sovelluskehyksen avulla rakennetut sovellukset ovat niin sanottuja hybridisovelluksia. Niitä ei kirjoiteta alustan omaa ohjelmointikieltä käyttäen, vaan internetohjelmoinnissa useimmiten käytössä olevia tekniikoita, kuten HTML5, CSS3 ja JavaScript, hyödyntäen. Tämän jälkeen sovellus käännetään sovelluskehyksen avulla kohteena olevan mobiilikäyttöjärjestelmän käännöstyökaluilla puhelimeen asennettavaksi sovellukseksi.

3.3 Adobe PhoneGap Build -pilvipalvelu

Adobe PhoneGap Build on Adoben hallinnoima pilvipalvelu, joka on rakennettu Adobe PhoneGap -alustan avulla rakennettujen sovellusprojektien kääntämiseen natiiveiksi sovelluksiksi. Adobe PhoneGap Build -pilvipalvelun avulla sovelluskehittäjä voi kääntää sovelluksen kaikille Adobe PhoneGapin tukemille alustoille yhdellä kertaa. Tavallisesti tämä prosessi pitää suorittaa komentoriviltä jokaiselle alustalle erikseen tai kirjoittaa skripti, joka ajaa nämä komennot automaattisesti.

Sovellus pitää tämän jälkeen vielä ladata kuvassa 2 näkyvästä Adobe PhoneGap Build -pilvipalvelusta omalle koneelle ja asentaa kehityslaitteena toimivalle älypuhelimelle käsin. Tämän takia kehitystyön kannalta on nopeampaa suorittaa tarvittava käännös paikallisesti esimerkiksi XCoden tai Android Studion avulla.



Kuva 2. Käännetyt versiot näkyvät valmiina ladattavaksi Adobe PhoneGap Build -palvelussa.

Käännösprosessi kannattaa siis tehdä kehitysvaiheessa paikallisesti, koska tällä säästytään siltä, että sovellus pitää ensin manuaalisesti ladata Adobe PhoneGap Build -palveluun, lisätä tarvittavat provisiointiprofiilit Apple iOS -alustalle ja odottaa sen aikaa kun Adobe PhoneGap Build kääntää sovellusta taustalla. Tämän jälkeen kuitenkin joudutaan vielä manuaalisesti tekemään asennus kehityslaitteena toimivalle puhelimelle, joten palvelun hyöty kehityksen aikana ei ole kovin hyvä.

Yksi tapa asentaa Adobe PhoneGap Build -pilvipalvelussa käännetty Google Android -sovellus on ladata käännetty sovelluksen asennuspaketti (APK, tiedostoformaatti), joka sitten asennetaan Android Debug Bridge -komentorivityökalun avulla älypuhelimeen. Apple iOS -alustalle asennusta varten tarvitaan vielä erikseen provisointitiedosto. (Android Debug Bridge 2014; iOS Dev Center 2014.)

4 Sovellukset Apache Cordova- tai Adobe PhoneGap -sovelluskehystä käyttäen

Adobe PhoneGap -sovelluksen voidaan katsoa koostuvan useammasta eri kerroksesta. Sovellusprojektin alussa on tärkeää arvioida, onko kaikki halutut ominaisuudet mielekästä ja mahdollista toteuttaa Adobe PhoneGap -sovelluskehyksen avulla. Projektin pi-

tuudesta ja tuetuista alustoista riippuen on harvemmin järkevää kirjoittaa omia alustakohtaisia ominaisuuksia. Ne joudutaan aina kirjoittamaan alustakohtaisesti alustan omalla kielellä ja työkaluilla. (Cordova 2014.)

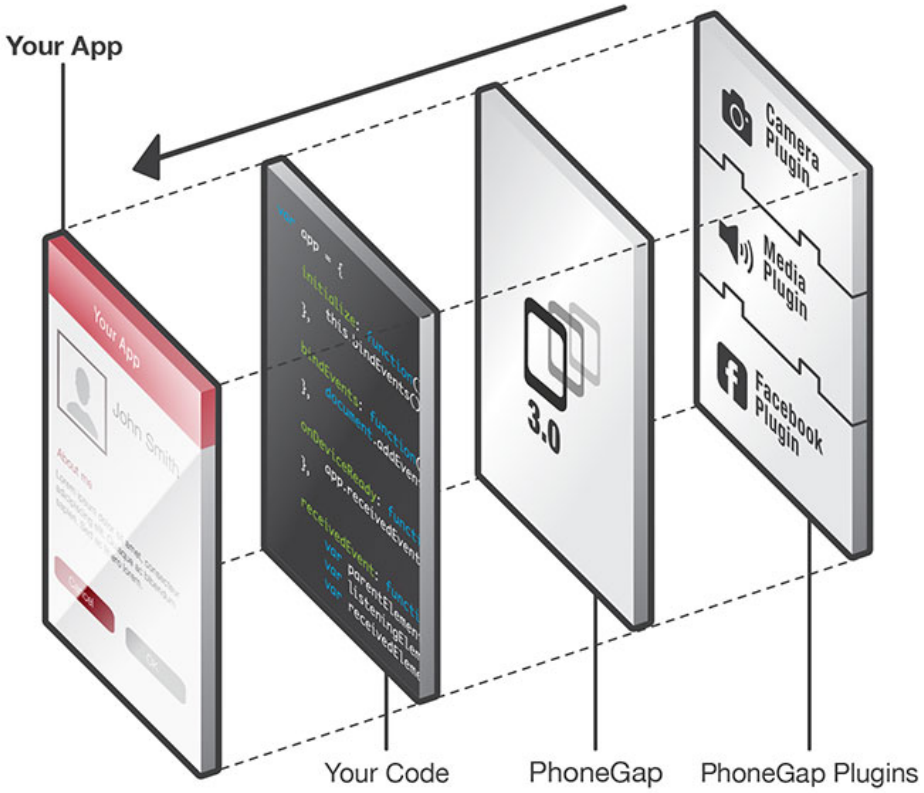
Taulukossa 2 nähdään Adobe PhoneGap -sovelluskehyksen tukemat laitetason ominaisuudet eri alustoilla. Vain vanhemmanmallisilla Apple iPhone -laitteilla on puutteita tuessa. Google Android on listattu yhtenä versiona, ja tällä tarkoitetaan aina uusinta Androidin versiota, joka tällä hetkellä on Google Android 4.4 KitKat.

Taulukko 2. Adobe PhoneGap sovelluskehyksen tukemat laiteason ominaisuudet eri alustoilla (Cordova 2014).

	iPhone / iPhone 3G	iPhone 3GS ja uudemmat	Android	Windows Phone 7 + 8
Kiihtyvyysanturi	x	x	x	x
Kamera	x	x	x	x
Kompassi	-	x	x	x
Kontaktit	x	x	x	x
Tiedostot	x	x	x	x
Geolokaatio	x	x	x	x
Media	x	x	x	x
Verkkoyhteydet	x	x	x	x
Ilmoitukset (Hälytys)	x	x	x	x
Ilmoitukset (Ääni)	x	x	x	x
Ilmoitukset (Värinä)	x	x	x	x
Tallennustila	x	x	x	x

Yksi Adobe PhoneGap -sovelluksenkehyksen suurimmista eduista muihin vastaaviin alustoihin, kuten esimerkiksi AppGyver Steroidsiin verrattuna on, että PhoneGap-projektin ympärille on ehtinyt vuosien myötä muodostua laaja sovelluskehittäjien joukko, joka kehittää avoimen lähdekoodin periaatteilla erilaisia liitännäisiä (plugin) Adobe PhoneGap -sovelluskehyksen ympärille. Tämän ansiosta Adobe PhoneGap -sovellukseen on usein saatavilla hieman harvinaisempiakin liitännäisiä, kuten esimerkiksi liitännäinen Applen kehittämän sisätilapaikannusjärjestelmän, iBeacon, paikannuslaitteille.

Kuvassa 3, näkyy miten Apache Cordova- tai Adobe PhoneGap-sovellus rakentuu eri kerroksista. Kerroksista kaksi ensimmäistä ovat ne, joiden parissa suurin osa kehittäjän ajasta menee Adobe PhoneGap -sovellusta ohjelmoidessa. Sovelluksen logiikka kirjoitetaan JavaScript-kielellä, ja siinä käytetään hyväksi Adobe PhoneGapin tarjoamia ohjelmointirajapintoja, joilla päästään käsiksi laitealustan tarjoamiin ominaisuuksiin, kuten esimerkiksi puhelimen muistiin tai natiivin Facebook-sovelluskehityskirjaston käyttämiseen. (Cordova 2014; LeRoux 2013.)



Kuva 3. Apache Cordova- ja Adobe PhoneGap -sovelluksen rakenne (LeRoux 2013).

Adobe PhoneGap -sovellukset koostuvat siis HTML5-, CSS3- ja JavaScript-tekniikoilla toteutusta lähdekoodista. HTML5 ja CSS3 määrittelevät sen, miltä sovellus näyttää ja nykyisin myös sen, miltä erilaiset animaatiot näyttävät sovelluksen sisällä. JavaScriptillä taas kirjoitetaan sovelluksen logiikka ja se, miten sovellus reagoi käyttäjän antamaan palautteeseen. Palaute voi tässä tapauksessa olla esimerkiksi kosketus kosketusnäytöllä, ele kosketusnäytöllä, näppäimistöllä annettu syöte tai kameralla otettu kuva. JavaScriptiä käytetään myös, kun halutaan käyttää mobiilialustan laitetason ominaisuuksia eri ohjelmointirajapintojen avulla, joita sovelluskehys antaa kehittäjälle käytettäväksi. (Cordova 2014.)

Sovelluskehittäjä tarvitsee Apple iOS -alustalle ohjelmoidessaan Apple MacOS X -käyttöjärjestelmällä varustetun tietokoneen ja Apple Xcode -kehitystyökalut. Tämä tarkoittaa, että Apple iOS -alustalle kehitettäessä on hankittava jokin Applen tietokoneista. Uusin Apple MacOS X -käyttöjärjestelmän versio on Apple MacOS X 10.9 Mavericks. Tässä kohtaa on hyvä huomata, että markkinoilla on käytettynä saatavilla Apple-laitteita, jotka eivät ole Intel x86 -pohjaisia, ja niille ei saa uusinta MacOS X -versiota eikä sitä myöten uusinta versiota Xcode-sovelluskehitysympäristöstä.

Google Android -alustalle kehitysympäristönä voi käyttää Windows-, Linux- tai MacOS X-käyttöjärjestelmiä. Google tarjoaa myös hyvät kehitystyökalut ilmaiseksi. Androidin kanssa yleisempiä kehitysympäristöjä ovat Android Studio ja Eclipse. Tämän insinöörityön prototyyppisovellukseen valittiin kehitysympäristöksi Android Studio.

4.1 HTML5-kuvauskieli

HTML5 on HTML-kuvauskielen uusin versio. Se tarjoaa monia uudistuksia edellisiin versioihin nähden, ja HTML4:ään verrattuna siitä ei ole erikseen eri tason määritelmiä. HTML5 tukee monia uusi elementtejä edellisiin HTML-kuvauskielen versioihin nähden. Näistä uusista ominaisuuksista mielenkiintoisimpia ovat audio, canvas, geolocation ja localstorage. Sovelluskehityksen kannalta HTML5 on se versio, jolle suurin osa web-sovelluksista kirjoitetaan. (HTML & CSS 2014; HTML Living Standard 2014.)

HTML5-standardoinnissa on mukana useita tahoja W3C:ssä (World Wide Web Consortium), jonka Tim Berners Lee perusti vuonna 1994. W3C tarkoituksena on taata WWW:n (World Wide Web) kehitys ja kasvaminen tulevaisuudessa. W3C:n lisäksi on muita yhteenliittymiä, jotka toimivat webin kehittämisen parissa. Niistä yksi merkittävin on Web Hypertext Application Technology Working Group eli WHATWG.

WHATWG on kasvava yhteisö yksityishenkilöitä ja eri alan yrityksissä toimivia ammattilaisia, jotka ovat kiinnostuneet edistämään HTML-kuvauskieltä ja web-sovellusten kehittämisessä tarvittavien rajapintojen kehitystä. WHATWG perustettiin vuonna 2004 W3C-työpajan jälkeen. Perustajajäseniä tuli sellaisista yrityksistä kuin Apple, Mozilla Foundation ja Opera Software. Pääasiallinen syy WHATWG:n muodostamiseen oli perustajäsenien huoli siitä suunnasta, mihin HTML-kuvauskielen kehitys oli menossa W3C:n suunnitelmissa. (Frequently Asked Questions (FAQ) about the future of XHTML 2009; HTML Living Standard 2014.)

Taulukossa 3 näkyvät yleisimmät käytössä olevat selainmoottorit ja niitä käyttävät selaimet. Näistä suurin osa tukee kattavasti HTML5-kuvauskielen määritteitä. HTML5-tuki on erinomainen myös mobiililaitteissa, ja uudemmat älypuhelimet tukevat HTML5:n uudempiakin ominaisuuksia, kuten Canvas-elementtiä. (HTML Living Standard 2014.)

Selainmoottori	Missä selainmoottori on käytössä
Blink	Chromium/Chrome (28+), Opera (15+) ja Yandex
Gecko	Kaikissa Mozilla Softwaren ohjelmistoissa, esimerkiksi Firefox
KHTML	Konqueror
Presto	Opera ja Opera Mobile
Tasman	Internet Explorer 5+ Mac OS X
Trident	Internet Explorer, Windows Phone 8
WebKit	Safari (työpöytä ja mobiiliversiot), Google Chrome (ennen versiota 28), Android-selain

Taulukko 3. Yleisimpien internetselainten käyttämät selainmoottorit.

4.2 CSS3-tyylimääritteet

CSS eli Cascading Stylesheets ovat tyyliohjeita, jotka määrittelevät, miltä selainmoottorin rakentama DOM-rakenne näyttää selaimessa. W3C määrittelee ja ylläpitää CSS-spesifikaatiota, ja uusin CSS-versio on CSS3. Tämä versio lisäsi monia uusia ominaisuuksia, joita tukevat nykyään kaikki nykyaikaiset selainmoottorit lähes täydellisesti.

CSS-tyylit määrittyvät nimensä mukaisesti porrastetusti. Uusin luettu CSS-tyyliohje yli kirjoittaa aina edellisen tyyliohjeen, poikkeuksia tosin on, kuten esimerkiksi ”!important”-määritteen käyttäminen. Tämä tekee CSS-tyylimäärittelyiden kirjoittamisesta tiiviimpää, sillä samoja määrittelyitä ei tarvitse kirjoittaa uudestaan, vaan ne periytyvät, ja tarvittaessa nämä määrittelyt voidaan ylikirjoittaa. (Souders 2007.)

CSS3 toi mukanaan monia uusia ominaisuuksia, joiden avulla pystytään rakentamaan paremmannäköisiä selainmoottorin rakentamia DOM-rakenteita. Yksi käytetyimmistä on CSS3-animaatiot, jotka mahdollistavat sen, että ei ole enää tarpeen käyttää esimeriksi JavaScriptiä DOM-elementin animaatioon (Kinnunen 2014). Tämän insinöörityön aiheena olevan prototyyppisovelluksen tapauksessa sovelluksen käyttöliittymä hyödyntää kokonaisuudessaan CSS3-tyylejä ja käyttää kattavasti CSS3-animaatioita.

Yksi hyödyllisimmistä CSS3-ominaisuuksista ovat laitetasolla kiihdytetyt CSS-tyylimääritteet, joita on esimerkiksi transform. Transform ei vielä ole osa virallista CSS3-määrittelyä, mutta sillä on laaja tuki uudemmissa selainmoottoreissa ”vendor prefixien” eli selainmoottorikohtaisten määritteiden kautta. Tähän CSS-tyylimääritteeseen kuuluu monia

eri parametreja, joiden avulla DOM-elementtejä voidaan siirtää erilaisiin asentoihin. CSS3 transform -tyylimääritteellä voidaan esimerkiksi nopeuttaa osan CSS3-animaatioiden tai piirtojen sujuvuutta. (Kinnunen 2014.)

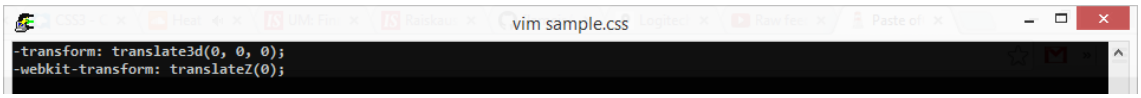
Adobe PhoneGap -pohjaisten sovellusten ulkoasu määritellään CSS-tyyliohjeita käyttäen. Tämä tekee ulkoasun muokkaamisesta hyvin helppoa sellaisellekin kehittäjälle, jolla ei ole alustakohtaista kokemusta.

4.3 Selainmoottorin piirto- ja reflow-tapahtumat

Suorituskyky on yksi asia, joka yleensä erottaa natiivin sovelluksen hybridisovelluksesta. Natiivissa sovelluksessa kaikki piirrot tapahtuvat laitetasolla kiihdytetysti ja suorituskyky pysyy jo tästäkin syystä parempana kuin hybridisovelluksessa. Hybridisovelluksessa päästään kuitenkin varsin toimivaan ja natiivin kaltaiseen nopeuteen kun sovelluksen ohjelmoinnissa otetaan huomioon, miten selainmoottorit suorittavat piirrot. (Kinnunen 2014; Souders 2007.)

Piirtotapahtumien seuraaminen onnistuu myös Adobe PhoneGap -sovelluksissa Safari-selainta käyttämällä. Tämä tosin tarkoittaa sitä, että vain iOS-alustalle tulevien sovellusten piirtotapahtumien määrää voidaan helposti seurata selaimen debug-työkaluilla.

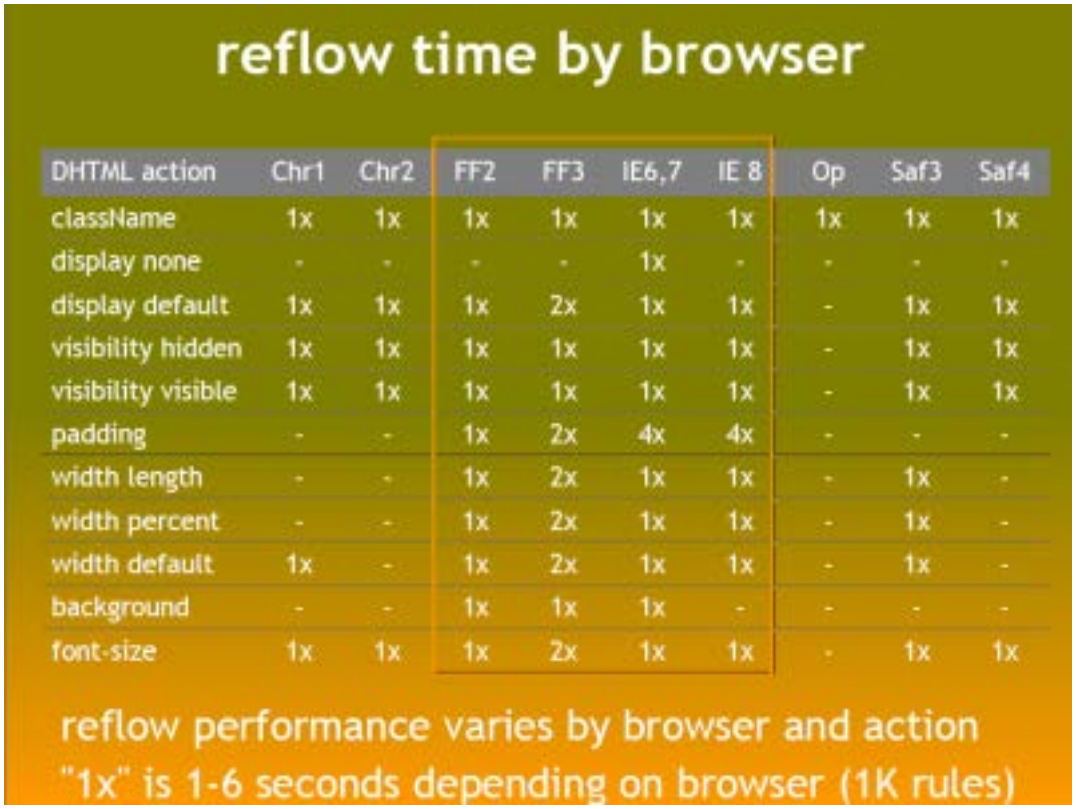
Selainmoottori voidaan hyvin yksinkertaisilla kuvassa 4 esitellyillä CSS-tyylimääritteillä pakottaa piirtämään DOM-elementti laitetasolla kiihdytetysti. Tämä ”-webkit-transform: translate3d(0, 0, 0); ” tai ”-webkit-transform: translateZ(0);” CSS-tyylimäärite pakottaa selainmoottorin hyödyntämään laitteen näytönohjainpiiriä (GPU, Graphics Processing Unit) DOM-elementin piirtämiseen. (Kinnunen 2014.)



Kuva 4. Kaksi eri CSS-tyylimääritettä, jolla selainmoottori piirtää DOM-elementin laitetasolla-kiihdytetysti.

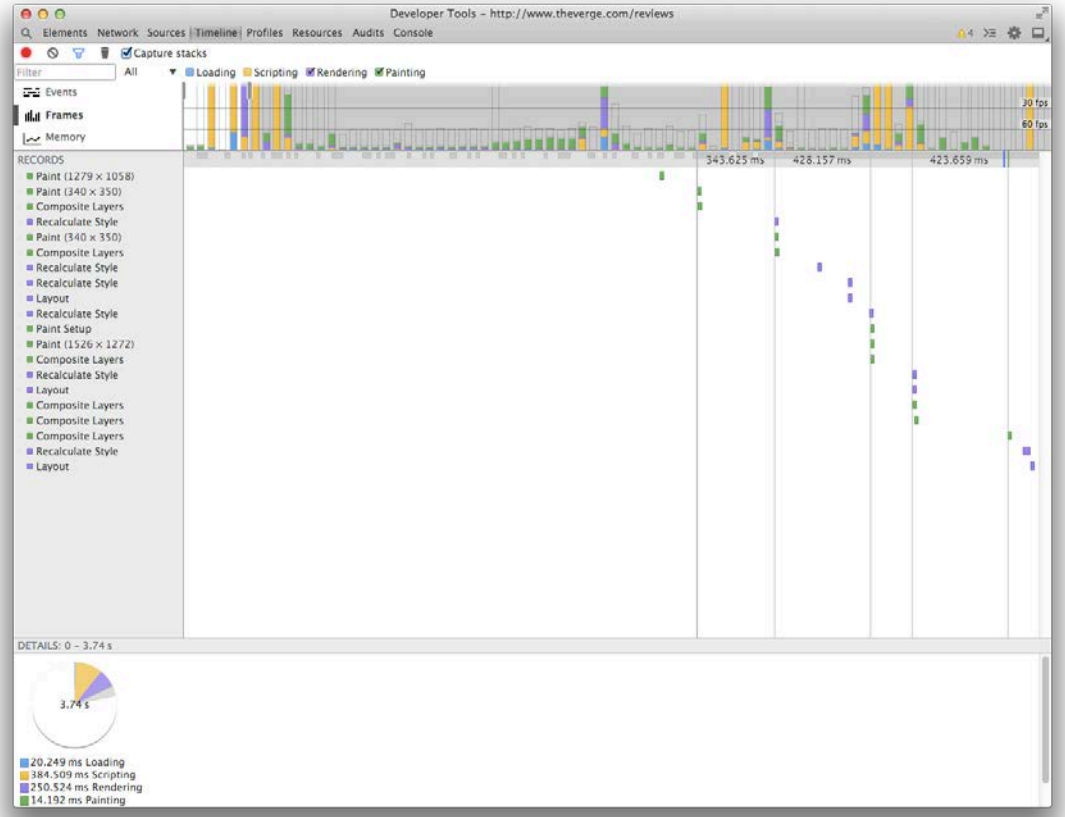
Selainmoottori piirtää DOM-rakenteen asteittain, mikä näkyy käyttäjille siten, että sivusto näyttää tulevan näkyviin nopeammin, vaikka kaikkea ei vielä ole ehditty piirtää. Selain-

moottori suorittaa useita piirtoja, kunnes DOM-rakenne on valmis. Piirto- ja reflow-tapahtumien määrään vaikuttavat DOM-rakenteen CSS-tyylimääritteet ja DOM-rakenteen muokkaaminen jälkikäteen ohjelmallisesti. Kuvassa 5 näkyy, kuinka paljon eri selaimissa eli niiden selainmoottoreissa reflow-tapahtumat vaikuttavat suorituskykyyn. (McQuade 2014; Simon 2014; Souders 2007.)



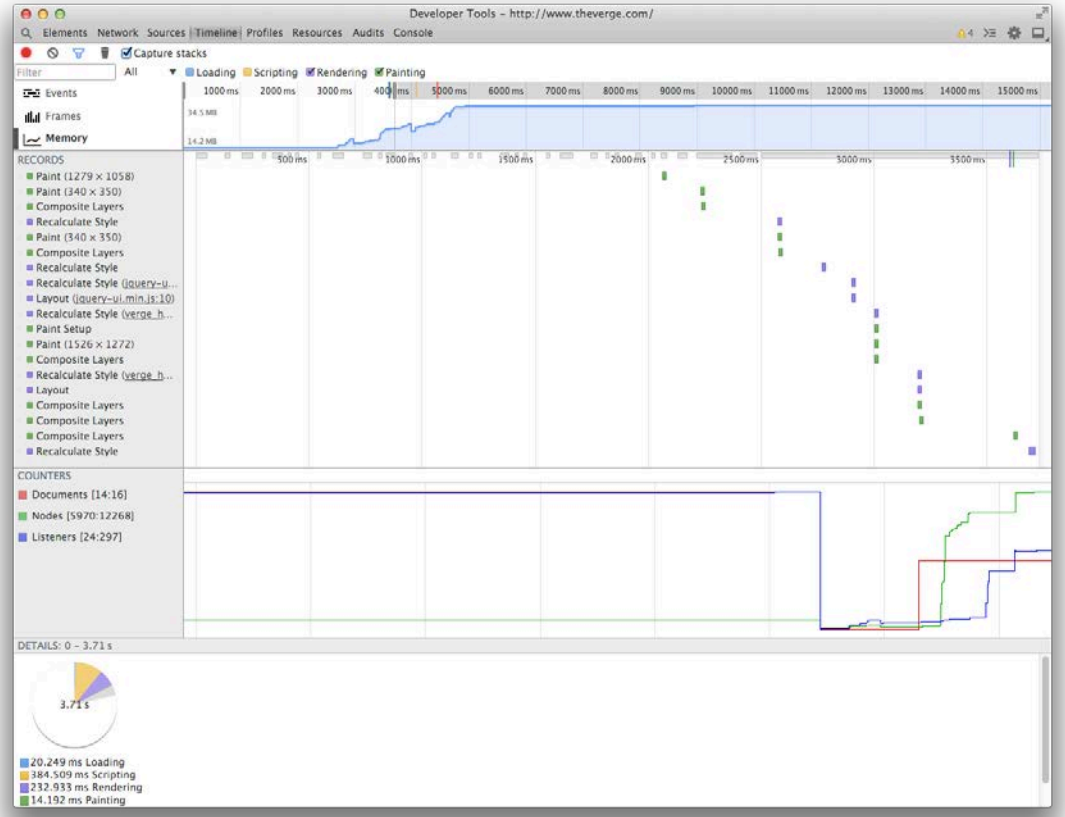
Kuva 5. Eri CSS-tyylimääritteiden vaikutus selainmoottoreiden reflow-nopeuteen (McQuade 2014).

DOM-rakenteen muokkaus esimerkiksi CSS-tyylimääritteillä DOM-elementin luokkaa vaihtamalla tai JavaScriptin avulla aiheuttaa sen, että selainmoottori suorittaa yhden tai useamman piirron. Piirtojen ja reflow-tapahtumien määrä vaikuttaa hyvin suoraan suorituskykyyn, ja niiden määrä olisikin hyvä pitää mahdollisimman pienenä. Kuvassa 6 näkyy tavanomaisen ison verkkosivuston piirtotapahtumien lomittuminen. CSS-tyylimäärittelytiedostot kannattaa myös pitää dokumentin yläosassa koska selain lataa DOM-puussa vastaan tulevat ulkopuoliset resurssit järjestyksessä. Rinnakkaisia GET-pyyntöjä useimmat selainmoottorit suorittavat enintään noin kuusi kerrallaan. (McQuade 2014; Simon 2014; Souders 2007.)



Kuva 6. Piirtotapahtumien lomittuminen Google Chrome -selaimessa.

Selainmoottorit myös jakavat DOM-rakenteen lohkoihin, ja näitä lohkoja selainmoottori piirtää taustalla valmiiksi. Tästä syystä selaimet pystyvät piirtämään isohkojakin sivustoja kaatumatta. Sivuston DOM-rakenteen lohkominen auttaa myös muistinkäytössä. Kuvassa 7 nähdään selainmoottorin varaaman muistinmäärän kasvu, kun sivuston piirtämisessä edetään. Näitä kaikkia asioita selainmoottoreiden kehittäjät kohtaavat päivittäin, ja niiden optimoiminen tarkoittaa myös parempaa suorituskykyä mobiiliselaimissa. Ikävä kyllä, parannuksia harvemmin saadaan takautuvasti WebView-selainpuolelle. (Kinnunen 2014; McQuade 2014.)



Kuva 7. Webkit-selainmoottorilla varustetun selaimen muistinkäyttö.

Osaa näistä selainmoottorin piirtotapahtumista voidaan nopeuttaa CSS transform -määritteen avulla, jolloin elementti on laitetasolla kiihdytetty. Tämä lähestymistapa on hyvä oikein käytettynä, mutta kääntöpuolena on muistin käytön kasvaminen. Muistia ei ole järin paljon käytössä esimerkiksi vanhemmissa Android- tai iOS-laitteissa. Toisin sanoen, ei ole käytännössä mahdollista tehdä sivustoa siten, että kaikki elementit olisivat laitetasolla kiihdytettyjä, koska laitteiden muistimäärät ovat rajoitettuja. (Kinnunen 2014; McQuade.)

4.4 JavaScript- ja ECMAScript-ohjelmointikielet

JavaScript on alkujaan Brendan Eichin NetScape-internetselainta varten kehittämä prototyyppipohjainen dynaaminen ohjelmointikieli. Alun perin kielen tarkoitus oli olla kevyempi vaihtoehto Java-ohjelmointikielelle. JavaScriptin ensimmäinen versio julkaistiin vuonna 1995, ja nykyään JavaScript on laajimmalle levinnyt ohjelmointikieli selaimissa. (Crockford 2008; Seibel 2009:135–150.)

Alun perin JavaScript-moottorit tulkkasivat lähdekoodia rivi kerrallaan ja käsittelivät sitä vastaavasti. Tämä on huomattavasti hitaampaa kuin käännetyn koodin suorittaminen. Kuitenkin kääntämisen suorittaminen olisi vienyt ensimmäisellä kerralla huomattavasti enemmän aikaa, ja tästä syystä alun perin on päädytty käyttämään tulkkipohjaista lähestymistapaa. Nykyään JavaScript-moottorit ovat kehittyneet niin paljon, että Just-in-time-kääntäminen on jo arkipäivää. (Crockford 2008; JavaScript technologies overview 2014.)

JavaScript-ohjelmointikielen tulkkaamisen ja kääntämisen hoitaa selaimen JavaScript-moottori. Näitä erilaisia JavaScript-moottoreita on useita, mutta tällä hetkellä erittäin laajassa käytössä muuallakin kuin vain selaimissa on Googlen kehittämä V8 JavaScript Engine. Se on erittäin nopea, ja se hyödyntää Javastakin tuttua Just-in-time-kääntämistä. Googlen V8 kääntää JavaScript-koodin konekieliseksi koodiksi, ja tuettuina arkkitehtuureina ovat IA-32, x86-64, ARM ja MIPS. Näistä x86-64 on käytössä nykyisissä PC-tietokoneissa, mukaan lukien Applen PC-tietokoneet. ARM on puolestaan lähes ainoa arkkitehtuuri, jota käytetään älypuhelimissa. (Chrome V8 2014.)

ECMAScript on standardisoitu kielimäärittely, johon JavaScript pohjautuu. Ecma International määrittelee ECMAScriptin ECMA-262-spesifikaatiossa ja ISO/IEC 16262 -standardissa. Eri JavaScript-moottoritoteutukset noudattavat tätä määrittelyä, mutta ECMA-262 ei sisällä kuitenkaan kaikkia DOM-elementtien kanssa tarvittavia ohjelmointirajapintoja. ECMAScript-määrittely sisältää yleistasolla:

- ohjelmointikielen syntaksin, kuten esimerkiksi avainsanat, suoritusrakenne, olioliteraalit
- virheidenhallintamekanismit, kuten esimerkiksi throw, try/catch
- tyypit, kuten esimerkiksi number, string, function, boolean, object
- globaalin olion; selaimissa tämä on toteutettu window-oliona, mutta ECMAScript määrittelee vain ohjelmointirajapinnan, ei itse toteutusta
- prototyyppipohjaisen perintämekanismin
- sisäänrakennetut oliot ja funktiot, kuten esimerkiksi JSON ja Math
- strict-määrittelyn. (Crockford 2008; Standard ECMA-262 Draft for 6th edition 2014.)

ECMAScriptin yleisin tuettu versio on ECMAScript 5 (ES5), mutta ECMAScript 6 (ES6) on jo hyvin lähellä valmistumista, ja siitä tulee seuraava käyttöön tuleva versio. Ongelma

tosin on se, että vanhemmat selaimet eivät tue suoraan uutta versiota kokonaisuudessaan. Tämän ongelman ratkaisuksi on kehitetty niin sanottuja välikääntäjiä, jotka kääntävät ES6-koodin ES5 JavaScript -moottoreiden ymmärtämään muotoon. (Standard ECMA-262 Draft for 6th edition 2014.)

ECMAScript 6 -määrittely tuo mukanaan monia kehittäjien toivomia ominaisuuksia, kuten vakioarvot funktioiden parametreille ja parannetun olioliteraalien syntaksin. ECMAScript 6:n odotetuimmat ominaisuudet ovat kuitenkin moduulit, sisällytys (import) ja luokkamäärittelyt. Näiden avulla JavaScript-ohjelmien rakenteen selkiyttäminen on huomattavasti helpompaa eikä tarvita erillistä JavaScript-kirjastoa, kuten esimerkiksi Require.js tämän ongelman selvittämiseen. (JavaScript technologies overview 2014; Standard ECMA-262 Draft for 6th edition 2014.)

Nykyinen suuntaus niin internet- kuin hybridimobiilisovelluksissa ovat niin sanotut single-page-app-sovellukset. Nämä niin sanotut SPA-sovellukset tarkoittavat sitä, että kaikki navigaatio tapahtuu JavaScript-sovelluksen kautta. Näin ei tehdä erillisiä sivunlatauksia ja pystytään luomaan enemmän perinteisen työpöytä- tai mobiilisovelluksen mielikuva käyttäjälle. (Aghassipour & Shajith 2014.)

5 Ember.js-JavaScript-sovelluskehys

Ember.js on avoimen lähdekoodin JavaScript-sovelluskehys. Se on tarkoitettu asiakaspäähän, eli se toimii loppukäyttäjän JavaScript-moottoria käyttäen. Ember.js on myös MVC-arkkitehtuurimallin kaltaisesti jaettu eri osiin. M eli mallikerros on sovelluksen tietorakenteen, V eli näkymäkerros on sovelluksen ulospäin näkyvä kerros ja C eli ohjainkerros pitää sisällään sovelluksen suoritusrakenteen. Perinteisiin MVC-pohjaisiin sovelluskehyksiin verrattuna, kuten esimerkiksi Ruby on Rails, Ember.js ei sisällä perinteistä MVC-arkkitehtuurimallin mukaista rakennetta, ja osa perinteisen MVC:n mukaisista asioista on toteutettu useissa ”kerroksissa”. (Aghassipour & Shajith 2014; Ward 2013.)

Ember.js-projekti alkoi kun SproutCore 2.0 -projektin sovelluskehittäjät Tom Dale ja Yehuda Katz päättivät nimetä sen Ember.js:ksi. Tämä tehtiin siksi, että sekaannus SproutCore 1.0 -projektin, joka ei ollut sovelluskehys, välillä häivenisi. Ember.js:n pääkehittäjinä

toimivat Tom Dale ja Yehuda Katz, joilla molemmilla on pitkä historia avoimen lähdekoodin projekteissa, kuten jQuery, Ruby on Rails ja SproutCore. (Mueller 2014; Ward 2013; Walker-Morgan 2011.)

Nykyään Ember.js-projektin ympärillä on laaja avoimen lähdekoodin kehittäjien ja käyttäjien joukko. Mukaan on liittynyt myös Yahoo, joka tulee käyttämään kaikissa tulevissa projekteissaan Ember.js-JavaScript-sovelluskehystä. Ison internetalan yrityksen tuki takaa myös jatkuvuutta, ja sen merkitys on kohtuullinen, kun eri yritykset arvioivat eri sovelluskehysten soveltuvuutta omassa liiketoimintaympäristössään. (Mueller 2014; Ward 2013; Walker-Morgan 2011.)

Ember.js on sovelluskehys nykyaikaisten ja kunnianhimoisten sovellusten kehittämiseen JavaScript-ohjelmointikieltä käyttäen. Jo ideologisella tasolla Ember.js eroaa monista muista JavaScript sovelluskehyksistä siinä, että se painottaa käytäntöä eikä konfiguraatiota. Ember.js-sovellus on järkevää rakentaa hyödyntäen olemissa olevia hyväksi todettuja käytäntöjä. (Ward 2013.)

Muita vastaavia JavaScript-projekteja ovat Backbone.js, AngularJS ja ReactJS. Angular.js-projektin taustalla on Google ja ReactJS:n taustalla toimii Facebookin Instagram-tiimi. Angular.js on enemmänkin kokoelma työkaluja, joiden päälle voi rakentaa oman käytäntönsä mukaisen sovelluskehyksen. Backbone.js oli ensimmäisiä MVC-mallin mukaisia JavaScript-sovelluskehyksiä, mutta nykyään sen käyttö on vähenemään päin. Maailmalla on paljon muitakin JavaScript single-page-app -sovelluskehyksiä, mutta edellä mainitut JavaScript-sovelluskehykset ovat yleisimmin käytössä. (Aghassipour & Shajith 2014; Ward 2013.)

Ember.js koostuu useista erilaisista hyväksi todetuista käytännöistä. Ember.js-sovelluksen tilaa ylläpitää reititin eli router. Handlerbars.js toimii sivupohjien tulkkina, mallikerros on sovelluksen keskiössä ja kaksisuuntainen datan sitominen on yksi Ember.js-sovelluskehyksen vahvuuksista. Kaksisuuntainen datan sitominen tarkoittaa sitä, että data voidaan päivittää niin näkymäkerroksessa kuin mallikerroksessa, ja muutokset heijastuvat myös toiseen suuntaan. Ember.js käyttää nykyisessä versiossaan vielä jQuery-kirjastoa. Tulevaisuudessa tästä riippuvuudesta luovutaan, ja on käyttäjän valittavissa, käyttääkö erillistä DOM-manipulointiin tarkoitettua kirjastoa. (Ward 2013.)

5.1 Yksinkertainen Ember-esimerkkisovellus

Ember.js-sovelluskehys tekee paljon asioita taustalla automaattisesti, ja tämä automaatio on yksi syy, miksi Ember.js nojaa vahvasti käytöntöihin. Käytäntöjen avulla voidaan automatisoida sellaisia asioita, jotka tulisivat tehtyä kehittäjän toimesta useita kertoja kehityksen aikana. Ember.js ei esimerkiksi tarvitse Ember.Router-elementtiä ollenkaan, vaan Ember.js luo taustalla tarvittavat reitit aina tarvittaessa. Tarvittaessa kehittäjä voi toki kirjoittaa omat Route-elementtinsä, mutta ohjelman suoritus ei pääty, vaikka tämä elementti ei olisi lähdekoodissa määriteltynä.

Kuvissa 8 ja 9 näkyy, miltä yksinkertaisen Ember.js-sovelluksen rakenne näyttää niin DOM-rakenteen puolelta kuin itse lähdekoodista. Pienintä mahdollista Ember.js-sovellusta varten riittää kuvassa 9 ja rivillä 1 nähtävä lähdekoodin osa. Rivit 3–5 ovat tässä esimerkkisovelluksen tapauksessa tarpeettomia, vaikkakin isommassa sovelluksessa juuri Ember.Router-komponentti on yksi tärkeimmistä osista Ember.js-sovelluksessa. Rivit 7–11 luovat yksinkertaisen mallitason. Tämä mallitason tieto lisätään DOM-puuhun Handlebars.js:n toimesta, joka on Ember.js-sivupohjien tulkki.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>Ember Starter Kit</title>
6    <link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/normalize/2.1.0/normalize.css">
7    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
8    <script src="http://builds.handlebarsjs.com.s3.amazonaws.com/handlebars-v1.3.0.js"></script>
9    <script src="http://builds.emberjs.com/tags/v1.5.0/ember.js"></script>
10   <script src="app.js"></script>
11 </head>
12 <body>
13
14   <script type="text/x-handlebars">
15     <h2> Welcome to Ember.js</h2>
16
17     {{outlet}}
18   </script>
19
20   <script type="text/x-handlebars" data-template-name="index">
21     <ul>
22     {{#each item in model}}
23       <li>{{item}}</li>
24     {{/each}}
25     </ul>
26   </script>
27 </body>
28 </html>
```

Kuva 8. Ember Starter Kit, minimaalinen Ember.js-pohjainen sovellus.

Kuvassa 8 on kaksi erilaista Handlebars.js-sivupohjaa. Riveillä 14–18 näkyy koko sovelluksen sivupohja, ja riveillä 20–26 näkyy IndexRouten-sivupohja. Rivillä 17 näkyvä {{outlet}} tulostaa senhetkisen reitin sivupohjan tilalleen. Tässä esimerkissä on vain yksi reitti IndexRoute.

```
1  App = Ember.Application.create();
2
3  App.Router.map(function() {
4
5  });
6
7  App.IndexRoute = Ember.Route.extend({
8    model: function() {
9      return ['red', 'yellow', 'blue'];
10   }
11 });
```

Kuva 9. Ember.js-sovelluksen JavaScript-lähdekoodi.

Handlebars.js-sivupohjat sisältävät myös erilaisia kehittäjän avuksi olevia ominaisuuksia, kuten {{#each}}, joka näkyy kuvassa 8 rivillä 22. Tämä yksinkertainen komento iteroi mallin lävitse ja tulostaa jokaisen mallin omana DOM-elementtinään DOM-puuhun. Tässä tapauksessa malli on yksinkertainen JavaScript Array -tietorakenne, joka koostuu kolmesta eri string-elementistä: ’red’, ’yellow’ ja ’blue’.

5.2 Ember CLI -rakennustyökalu

Monet nykyiset avoimen lähdekoodin sovelluskehysprojektit hyödyntävät erilaisia rakennustyökaluja. Näiden työkalujen avulla voidaan automatisoida tiettyjä yleisempiä tehtäviä, jotka muuten jouduttaisiin suorittamaan käsin. Ember.js:n tapauksessa yksi tälläinen rakennustyökalu on Ember CLI. Sen avulla voidaan luoda esimerkiksi uusi Ember.js-sovelluspohja, missä on valmiina kaikki tarvittavat tiedot. Näin kehittäjän ei tarvitse itse ladata niitä internetistä ja varmistaa manuaalisesti, että kaikki asetukset ovat oikein. Yleisimmät Ember CLI -rakennustyökalun komennot näkyvät, tummennettuna alla olevassa listauksessa:

- **ember** - tulostaa kaikki Ember CLI:n käytettävissä olevat komennot
- **ember new <sovelluksen-nimi>** - luo Ember.js projektin kansion, joka on annettu parametrina ember new -komennolle
- **ember init** - generoi Ember.js-projektipohjan senhetkiseen kansioon
- **ember build** - rakentaa sovelluspohjan tiedostot
- **ember server** - käynnistää palvelimen portissa 4200 (vakio)
- **ember generate <generoijan nimi> <parametrit>** - generoi templaten, viewin, modelin ja muita sovelluksen tarvitsemia osia annettujen parametrien mukaisesti. (Penner 2014.)

Ember CLI on erinomainen apu kehittäjälle, kun tarkoituksena on kehittää modulaarinen kokonaisuus, joka noudattaa parhaaksi todettuja sovelluskehityksen malleja. Ember CLI myös hyödyntää JavaScript ES6 -kääntäjää, jonka avulla on mahdollista kirjoittaa ES6-määritelmän mukaista lähdekoodia. ES6-kääntäjän tekee lähdekoodista AMD-yhteensopivat versiot, joita nykyiset ES5-yhteensopivat JavaScript-moottorit tukevat. (Penner 2014.)

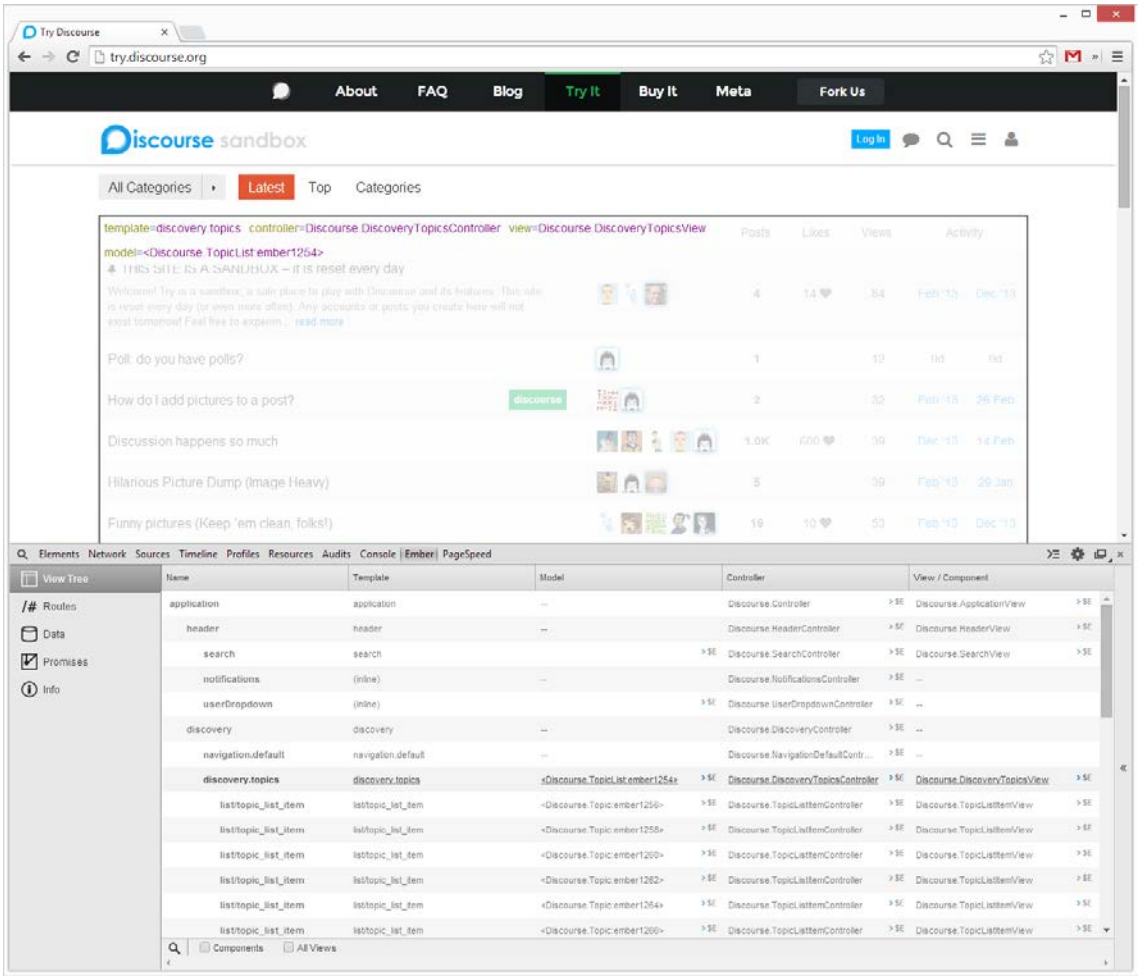
Kuten monet muutkin tässä insinöörityössä esitellyt työkalut ja työtavat, Ember CLI asennetaan myös node.js:n avulla npm-paketinhallintajärjestelmän kautta. Yksinkertaisimmillaan asennus onnistuu komennolla ”npm install -g ember-cli”. Tämä komento asentaa Ember CLI:n globaalisti järjestelmään, ja tämän jälkeen sovelluskehittäjän on mahdollista kutsua komentoriviltä ”ember”-komentoa ja sen eri alikomentoja. (Penner 2014.)

5.3 Ember-tools Chrome-lisäosa

Ember.js:ää hyödyntävien Apache Cordova- tai Adobe PhoneGap -sovelluskehyksen päälle rakennettujen mobiilisovellusten virheiden etsiminen (debuggaus) käy helpoiten Google Chrome -selaimen Ember-tools-lisäosaa käyttäen. Sovelluskehityksen alkuvaiheessa tämä työkalu on erittäin hyödyllinen, sillä sen avulla kehittäjä voi varmistua siitä, että sovelluksen perusrakenne on kunnossa ja asiat toimivat halutusti.

Kuvassa 10 näkyvät esimerkiksi kaikki Discourse-keskustelufoorumialustan senhetkisen näkymän view- eli näkymäkerroksen osat. Ember-tools antaa hyvin tarkat tiedot siitä,

mitä esimerkiksi kyseinen näkymä käyttää kontrollerinaan ja mikä näkymän malli on. Tämänkaltainen informaatio on hyödyllistä virheiden etsinnän kannalta, etenkin näin visuaalisessa muodossa. (Ward 2013.)



Kuva 10. Discourse-keskustelufoorumialusta Ember-tools kautta tarkasteltuna.

6 Adobe PhoneGap- tai Apache Cordova -pohjaisen sovelluksen vian etsintä

6.1 Apache Ripple -emulaattori

Apache Ripple on Google Chrome -selaimeen saatavilla oleva laajennus, minkä avulla voidaan emuloida monia eri Apache Cordova- tai Adobe PhoneGap -sovelluksessa käytettäviä puhelimen laitetason ominaisuuksia, kuten esimerkiksi paikkatietojen emulointi.

Tämä nopeuttaa kehitysprosessia merkittävästi, kun sovellusta ei tarvitse erikseen kääntää ja asentaa laitteeseen testausta varten. Tosin, kuten aina, lopullinen testaus tulee suorittaa oikealla laitteella. Emulaattori ei pysty emuloimaan kaikkia erikoistapauksia, joita puhelinsovelluksen kehityksen aikana tulee väistämättä vastaan. (Apache Cordova 2014; Apache Ripple 2014.)

6.2 Apache weinre -etäinspektori

Apache weinre eli WEb INspector REmote on selaimen etäinspektoriohjelma. Apache weinre -ohjelman avulla voidaan tarkastella, mitä tapahtuu puhelimen selaimessa tai WebView-komponentissa. Apache weinre hyödyntää node.js-alustaa, joka on rakennettu Google V8 JavaScript Enginen päälle. (Apache Weinre 2014.)

node.js on Google V8 JavaScript Enginen päälle rakennettu alusta, jonka päälle voidaan rakentaa erilaisia ohjelmistosovellutuksia. node.js-alustaa käytetään moniin eri tarkoituksiin, joista yksi on erilaisten komentorivityökalujen tarjoaminen. Näitä työkaluja varten on kehitetty paketinhallintajärjestelmä npm (node package manager), jonka avulla käyttäjän on helpompi hallita asennettuja node.js-paketteja. (Google V8 2014.)

6.3 Apple Safari- ja Google Chrome -selaimet vianetsinnän apuna

Apple iOS:n ja Apache Cordovan tai Adobe PhoneGapin yhdistelmän etuina on se, että Apache Cordova- tai Adobe PhoneGap -sovellusta pystytään tarkastelemaan suoraan Apple Safari -selaimen kautta. Apple Safari inspector on yksi tärkeimmistä työkaluista iOS-alustalle kehitetyn Apache Cordova- tai Adobe PhoneGap -sovelluksen vianetsintäprosessissa (debuggaus) virheiden selvittämiseksi. Apple Safari pohjautuu webkit-selainmoottoriin, ja käytössä on hyvät DOM-rakenteen ja JavaScript-ohjelmien debuggaustyökalut. Google Chrome kuitenkin tarjoaa samat työkalut, tosin näitä ei pysty hyödyntämään samalla tavalla suoraan Android-sovellusten kanssa ilman weinren kaltaista työkalua välissä. (Apple Safari Developer Guide 2014.)

Yksinkertaisimmillaan voidaan kutsua window.console.log-komentoa, jolla sovellus tulostaa selaimen JavaScript-konsoliin sen, mitä kehittäjä on syöttänyt log-metodille. JavaScript-ohjelmointikielen luonteen vuoksi, olion ja string-arvon tulostaminen samalla metodikutsulla ei onnistu, sillä JavaScript lisää olion string-muuttujan jatkeeksi.

Ember.js tarjoaa arvojen tulostamista varten omat lokifunktionsa, joita kutsumalla voidaan tulostaa sekä olio- että string-literaali. Kuvassa 11 näkyy pelkistetty esimerkki siitä miten Ember.js:n omaa Logger.log-metodia käytetään: metodille annetaan yksi tai kaksi parametria, riippuen siitä, halutaanko tulostaa vain olio tai arvo vai myös seliteteksti. (Apple Safari Developer Guide 2014.)

```
1 "use strict";
2
3 var valueOfOne = 1;
4
5 Ember.Logger.log('Value is', valueOfOne);
6
7 // OUTPUT
8 // "Value is: 1"
```

Kuva 11. Ember.Logger.log-metodin käyttäminen ja esimerkkituloste.

Virheiden etsintää lähdekoodissa auttaa huomattavasti kuitenkin niin sanottu askel- askeleelta -vian etsintä (step-by-step debuggaus). Tämä tarkoittaa sitä, että lähdekoodiin merkitään rivejä, joiden kohdalla suoritus pysähtyy ja eri muuttujien arvoja voidaan tarkastella siinä tilassa, missä ne ovat sillä hetkellä.

Usein pelkkä arvojen tulostaminen konsoliin ei tuota monimutkaisemmissa tapauksissa mitään loogista selitystä sille, miksi jokin asia ei toimi suunnitellusti. Askel- askeleelta -vian etsintä auttaa tämänkaltaisissa tapauksissa, kun sovelluskehittäjä saa paremman käsityksen siitä, mitä tapahtuu juuri virheellisen kohdan ympärillä eikä vain juuri sillä hetkellä ja siinä kohdassa. (Apple Safari Developer Guide 2014.)

Sovelluskehittäjän on myös mahdollista käyttää itse sovelluksen JavaScript-lähdekoodin seassa ”debugger;”-komentoa. Sen avulla avulla sovelluksen suoritus pysähtyy aina sille kohdalle, missä ”debugger;” on JavaScript-lähdekoodissa. Tämä on vastaava, kuin jos samalle riville olisi merkitty keskeytyspiste (break-point). (Apple Safari Developer Guide 2014.)

6.4 Kehitysprosessi

Insinöörityönä tehdyssä sovelluskehitysprojektissa ja lisätyn todellisuuden ratkaisuiden arvioinnissa käytetty työympäristö koostui seuraavista ohjelmistoista, sovelluskehyksistä ja kehitysympäristöistä:

- Apple MacOS X 10.9 Mavericks
- Apple Xcode 5
- Android Studio
- Adobe PhoneGap
- Adobe Photoshop CC
- Oh My Zsh
- Ember CLI
- Ember.js 1.5
- git.

Kehitysprosessi eteni asentamalla uusin Adobe PhoneGap -sovelluskehyspaketti npm:n avulla. Nykyään monet sovelluskehykset ja kirjastot käyttävät npm-paketinhallintasovellusta asentamiseen tai vaihtoehtoisesti git-versionhallintatyökalua.

Alkuvaiheessa kehitystyö etenee sujuvimmin siten, että rakennetaan itse sovelluspohja kuntoon. Adobe PhoneGap -komentorivityökalujen avulla voidaan generoida projektipohja, joka luo tarvittavat Xcode-projektitiedostot. Tämän jälkeen kirjoitin skriptin, joka kopioi tiedostot Ember.js-projektikansiosta itse Adobe PhoneGap -sovelluksen www-kansioon ja lisää tarvittavat viittaukset index.html-tiedostoon.

Tämän jälkeen voidaan projekti kääntää natiiviksi sovellukseksi Xcoden avulla. Xcode on myös se kehitysympäristö, jonka avulla projektiin liitetään sellaiset sovelluskehityskirjastot, joita osa liitännäisistä saattaa tarvita. Yleensä näiden liitännäisten konfigurointi tehdään myös Xcoden kautta.

Xcoden avulla käännetään seuraavassa vaiheessa sovelluksesta debugversio, jonka voi vaihtoehtoisesti suorittaa emulaattorissa tai oikealla laitteella. Oikealla laitteella testaaminen antaa paremmin kuvan lopullisesta suorituskyvystä, joten se on suositeltavampi

vaihtoehto näistä kahdesta. Apple Safari osaa etänä etsiä virheitä (debugata) myös oikealla laitteella suoritettavasta Adobe PhoneGap -sovelluksen debugversiosta, joten Apple iPhonen avulla virheiden etsintä (debuggaus) on sujuvaa.

Tätä samaa sykliä toistetaan aina tarvittaessa, kunnes kaikki ohjelmaan toivotut ominaisuudet on saatu ohjelmoitua. Yleisesti ottaen kehitysprosessi ei ole aivan yhtä sujuva, kuin jos kehitettäisiin vain yhdelle alustalle natiivia sovellusta. Toisaalta, koska voidaan hyödyntää JavaScript-, CSS3- ja HTML5-tekniikoita, on sovelluksen kehittäminen näiden tekniikoiden kanssa aikaisemminkin kehitystyötä tehneelle huomattavasti helpompaa, kuin opetella kokonaan uusi alusta ja tapa kehittää sovelluksia.

6.5 Prototyyppisovellus

Insinöörityön tuloksena syntynyt prototyyppisovellus hyödyntää Adobe PhoneGap -sovelluskehystä ja lisätyn todellisuuden Wikitude-liitännäistä. Sovelluksen kehittämiseen käytettiin HTML5-, JavaScript- ja CSS3-tekniikoita. Nämä edellä mainitut tekniikat, Adobe PhoneGap -sovelluskehys ja Wikitude-liitännäinen on esitelty tarkemmin luvuissa 3 ja 4.

Wikitude-liitännäisen tarvitsemat kuvatunnisteet lisätään Wikituden ylläpitämän pilvipalveluun, josta ladataan wtc-tiedosto, joka sisältää kuvatunnisteiden tiedot Wikitude-liitännäisen ymmärtämässä muodossa. Tämän kuvatunnistetiedon avulla Wikitude-liitännäinen osaa näyttää sovelluksessa lisätyn todellisuuden sisältöä, kun kamera tunnistaa kuvatunnisteen.

Kuvassa 12 näkyy yksinkertaisimmillaan AR.HtmlDrawable eli HTML-sisältökomponentti ja se, miten se sisällytetään Wikitudeen. AR.HtmlDrawablen kaikkia konfigurointiparametreja ei ole listattu tässä esimerkissä. Kuvassa 12 näkyvässä koodiesimerkissä ei ole koko listausta tarvittavasta koodista, kuten esimerkiksi this.worldLoaded-takaisinkutsuntafunktion määrittelyä. Wikitude GmbH on tuottanut kattavasti dokumentaatiota Wikitude PhoneGap -liitännäisestä, ja ongelmatilanteissa on järkevintä ensimmäisenä lukea ohjelmointirajapintadokumentaatiota lävitse. Toinen hyvä resurssi on stackoverflow-sivusto, jossa kehittäjä voi esittää eri tekniikoihin ja niiden käyttöön liittyviä kysymyksiä. (Wikitude SDK API Reference 2014.)

```
// Wikitude esimerkki
this.tracker = new AR.Tracker("assets/proto.wtc", {
    onLoaded: this.worldLoaded
});

var htmlContent = new AR.HtmlDrawable({
        uri: "assets/html-content-example-00.html"
    }, 1, {
    viewportWidth: 320,
    viewportHeight: 320,
    backgroundColor: "#FFFFFF",
    offsetX: 0,
    offsetY: 0,
    horizontalAnchor: AR.CONST.HORIZONTAL_ANCHOR.LEFT,
    verticalAnchor: AR.CONST.VERTICAL_ANCHOR.TOP,
    clickThroughEnabled: true,
    allowDocumentLocationChanges: false,
    onDocumentLocationChanged: function onDocumentLocationChangedFn(uri) {
        AR.context.openInBrowser(uri);
    }
});

var target_00 = new AR.Trackable2DObject(this.tracker, "target_00", {
    drawables: {
        cam: htmlContent
    }
});
```

Kuva 12. Wikituden sisällä olevan HtmlDrawable-elementin konfiguraatio.

Sovelluksen toiminta oli tarkoitukseensa nähden hyvä. Asiakasyrityksen tuntemus aihepiiristä kasvoi, ja prototyyppisovelluksen kehittämisestä saatua kehityskokemusta on hyödynnettävissä tulevaisuudessa eri lisätyn todellisuuden hankkeissa. Ei-tekniseltä kannalta tämäntyyppinen sovellus voisi toimia markkinoinnin keinona, mutta koska tämä vaatisi sen, että asiakas lataisi sovelluksen AppStore- tai Google Play Store -sovelluskauppapaikoista, mihinkään lyhytaikaiseen B2C-markkinointiin ei kannata tehdä tämän prototyyppisovelluksen kaltaista sovellusta. Markkinoilla on myös nykyään erilaisia valmiita sovellusalustoja, joiden päälle voidaan asiantuntijayrityksenä tuottaa mielekästä sisältöä. (Suvilaakso 2014.)

7 Tulokset ja loppupäätelmät

Insinöörityössä kävi hyvin ilmi se, että Apache Cordova- tai Adobe PhoneGap -pohjaisen sovelluksen tekeminen on tietyissä tapauksissa järkevä ratkaisu. Tämäntyyppinen hybridisovellus on yleensä paras ratkaisu, kun halutaan vielä testailla asioita. Prototyyppivaiheessa Apache Cordova- tai Adobe PhoneGap -sovelluskehyksen päälle rakennettu

sovellus voi olla oikea ratkaisu. Wikitude-lisäosa osoittautui kohtuullisen toimivaksi ratkaisuksi, kun halutaan tuoda lisätyn todellisuuden toiminnollisuuksia Apache Cordova- tai Adobe PhoneGap -sovelluskehysten päälle rakennettuihin sovelluksiin.

Monet asiat kuitenkin ovat sellaisia, joihin Apache Cordova- tai Adobe PhoneGap -alustat eivät tarjoa ratkaisuja. Sovelluksesta ei saa koskaan niin nopeata kuin natiivisovellus on, ja monet alustan tarjoamat palvelut eivät ole käytettävissä. Näistä palveluista voidaan mainita esimerkiksi natiivit karttasovellukset eri alustoilla. Natiiveja karttoja ei pystytä lisäämään Apache Cordova- tai Adobe PhoneGap -pohjaiseen sovellukseen, ja vaihtoehdoksi jää vain ulkopuolisten karttapalveluiden, kuten esimerkiksi Nokia HERE Maps, Mapbox tai Google Maps, käyttäminen.

Natiivin sovelluksen kehittäminen on kuitenkin pidempi prosessi, ja sitä kautta myös kalliimpaa asiakkaalle. Onkin erittäin tärkeää miettiä projektikohtaisesti, onko mobiilisovellus järkevintä toteuttaa natiivisovelluksena vai hybridisovelluksena. Mitä laajempi haluttu toiminnallisuus on, sitä todennäköisempää on, että ainoa järkevä vaihtoehto on käyttää mobiilikäyttöjärjestelmän natiiveja kirjastoja ja kehitysympäristöä. Tällä tavoin pystytään hyödyntämään jatkossakin kaikki mahdolliset laitteiston tarjoamat ominaisuudet.

Insinöörityössä löydettiin myös useita hyviä toimintamalleja sen suhteen, mitä kirjastoja ja työkaluja kannattaa ottaa käyttöön myös selainsovelluksiin. Insinöörityön aiheena olleet tekniikat kuitenkin pätevät suurilta osin myös selainsovellusten puolelle. Apache Cordova- ja Adobe PhoneGap -sovelluskehykset kuitenkin käyttävät runkonaan näitä hyväksi todettuja selaintekniikoita, kuten HTML5, CSS3 ja JavaScript.

Jatkon kannalta on varmasti järkevää panostaa resursseja esimerkiksi tässäkin insinöörityössä mainittuihin sovelluskehyksiin, kuten Ember.js tai AngularJS. Nämä nykyään jo huomattavan yleiset yhden sivun sovellukset (single-page-app) ovat kuitenkin selkeästi seuraava suunta, johon selainsovellukset ovat menossa. Osaavan kehittäjän käsissä tämäntyyppinen sovelluskehys säästää jatkossa aikaa vianetsinnässä, ja sovellukseen on paljon helpompi myöhemmin tuoda lisäominaisuuksia, koska testaus on paljon paremmin otettu huomioon jo sovelluskehyksiä suunniteltaessa.

Kokonaisuutena insinöörityö toi arvokasta tietoa siitä, missä niin sanottujen hybridimobiilisovelluskehysten kehitys menee tällä hetkellä. Samalla se vahvisti jo aiemmin todettuja hyviä ja huonoja puolia, joita hybridisovelluskehyksen kanssa sovellusta kehitettäessä

tulee vastaan. Suomalaisen AppGyver Oy:n kehittämä Steroids vaikuttaa erittäin mielenkiintoiselta alustalta, jo sen takia, että suorituskyky on lähes natiivilla tasolla, koska mobiilikäyttöjärjestelmän käyttöliittymäkirjasto on käytettävissä JavaScript-ohjelmointirajapinnan kautta. Tämä on varmasti yksi jatkotutkimuksen kohteista tällä alueella.

Lähteet

A vocabulary and associated APIs for HTML and XHTML. 2014. Verkkodokumentti. W3C. <http://www.w3.org/html/wg/drafts/html/CR/>. Luettu 12.3.2014.

Abrash, Michael. 1997. Graphics Programming Black Book. Verkkodokumentti. <http://www.drdobbs.com/parallel/graphics-programming-black-book/184404919>. Luettu 4.3.2014.

Aghassipour, Alexander & Chacko, Shajith. 2014. Verkkodokumentti. Techcrunch, AOL Inc. <http://techcrunch.com/2012/11/30/why-enterprise-apps-are-moving-to-single-page-design/>. Luettu 14.3.2014.

Android Debug Bridge. 2014. Verkkodokumentti. Google Inc. <http://developer.android.com/tools/help/adb.html>. Luettu 14.3.2014.

Apache Cordova Platform Support. 2014. Verkkodokumentti. Apache Software Foundation. <http://wiki.apache.org/cordova/PlatformSupport>. Luettu 17.4.2014.

Apache Cordova. 2014. Verkkodokumentti. Apache Software Foundation. <https://cordova.apache.org/>. Luettu 14.3.2014.

Apache Ripple. 2014. Verkkodokumentti. Apache Software Foundation. <http://ripple.incubator.apache.org/>. Luettu 14.3.2014.

Apache weinre. 2014. Verkkodokumentti. Apache Software Foundation. <http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html>. Luettu 14.3.2014.

Appel, Arthur. 1968. Some techniques for shading machine renderings of solids. Verkkodokumentti. <http://graphics.stanford.edu/courses/Appel.pdf>. 1968. Luettu 2.3.2014.

AppGyver. 2014. Verkkodokumentti. AppGyver Oy. <http://www.appgyver.com/>. Luettu 10.4.2014.

Apple Safari Developer Guide. 2014. Verkkodokumentti. Apple Inc. <https://developer.apple.com/library/safari/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/Introduction/Introduction.html>. Luettu 14.3.2014.

Azuma, Ronald. 1997. A Survey of Augmented Reality. Verkkodokumentti. <http://www.cs.unc.edu/~azuma/ARpresence.pdf>. Luettu 12.2.2014.

Chrome V8. 2014. Verkkodokumentti. Google Inc. <https://developers.google.com/v8/>. Luettu 1.3.2014.

Creating applications with Sailfish Silica. 2014. Verkkodokumentti. Jolla Ltd. <https://sailfishos.org/sailfish-silica/>. Luettu 17.4.2014.

Crockford, Douglas. 2008. JavaScript: The Good Parts. California: O'Reilly Media.

CSS3. 2014. Verkkodokumentti. Mozilla Developer Network and individual contributors. <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3>. Luettu 16.3.2014.

Forsyth, David A. & Ponce, Jean. 2003. Computer Vision: A Modern Approach. New Jersey: Pearson/Prentice Hall.

Frequently Asked Questions (FAQ) about the future of XHTML. 2009. Verkkodokumentti. W3C. <http://www.w3.org/2009/06/xhtml-faq.html>. Luettu 1.3.2014.

Google Glass. 2014. Verkkodokumentti. Google Inc. <https://developers.google.com/glass/>. Luettu 10.2.2014.

HTML & CSS. 2014. Verkkodokumentti. W3C. <http://www.w3.org/standards/webdesign/htmlcss>. Luettu 10.3.2014.

HTML Living Standard. 2014. Verkkodokumentti. WHATWG. <http://www.whatwg.org/specs/web-apps/current-work/multipage/>. Luettu 14.3.2014.

HTML: The Living Standard A technical specification for Web developers. Verkkodokumentti. WHATWG. <http://developers.whatwg.org/>. Luettu 14.3.2014.

Image Targets. 2014. Verkkodokumentti. Qualcomm Vuforia. <https://developer.vuforia.com/resources/dev-guide/image-targets>. Luettu 10.3.2014.

iOS Dev Center. 2014. Verkkodokumentti. Apple Inc. <https://developer.apple.com/devcenter/ios/index.action>. Luettu 14.3.2014.

JavaScript technologies overview. 2014. Mozilla Developer Network and individual contributors. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview>. Luettu 20.3.2014.

JavaScript Web APIs. 2014. Verkkodokumentti. W3C. <http://www.w3.org/standards/webdesign/script.html>. Luettu 14.3.2014.

John Carmack joins Oculus VR. 2013. Verkkodokumentti. Oculus VR. <http://www.oculusvr.com/blog/john-carmack-joins-oculus-as-cto/>. Luettu 10.2.2014.

Kinnunen, Kimmo. 2014. Senior Software Engineer. Nvidia Helsinki Oy, Helsinki. FINHTML5-seminaari. 20.3.2014.

LeRoux, Brian. 2013. Verkkodokumentti. Adobe. <http://phonegap.com/blog/2013/06/20/coming-soon-phonegap30/>. Luettu 7.3.2014.

Madden, Lester. 2012. Wikitude Technology Spotlight. Verkkodokumentti. Augmented Planet. <http://www.augmentedplanet.com/2012/09/wikitude-technology-spotlight/>. Luettu 2.2.2014.

McQuade, Bryan. 2014. Capturing and analyzing browser paint events using Page Speed Activity. Verkkodokumentti. Google Inc. <https://developers.google.com/speed/articles/browser-paint-events>. Luettu 16.3.2014.

Mueller, Samuel. 2014. Ember at Yahoo. Verkkodokumentti. <http://slid.es/sjmueller/ember-yahoo>. Luettu 13.3.2014.

Packaged Web Apps (Widgets) - Packaging and XML Configuration (Second Edition). 2014. Verkkodokumentti. W3C. <http://www.w3.org/TR/widgets/>. Luettu 14.3.2014.

Penner, Stefan. 2014. Ember CLI. Verkkodokumentti. <http://iamstef.net/ember-cli/>. Luettu 13.4.2014.

Seibel, Peter. 2009. Coders at Work. New York: Apress.

Simon, Lindsey. 2014. Minimizing browser reflow. Verkkodokumentti. <https://developers.google.com/speed/articles/reflow>. Luettu 17.3.2014.<

Souders, Steve. 2007. High Performance Web Sites: Essential Knowledge for Front-End Engineers. California: O'Reilly Media.

Standard ECMA-262 Draft for 6th edition. 2014. Verkkodokumentti. ECMA International. <http://wiki.ecmascript.org/lib/exe/fetch.php?id=harmony%3Aspecification_drafts&cache=cache&media=harmony:working_draft_ecma-262_edition_6_04-05-14.pdf>. 2014. Luettu 10.4.2014.

Suvilaakso, Ville, Senior Developer, Kuubi Visual Productions Oy, Helsinki. Keskustelu 1.4.2014.

Svenn, Antti. 2014. Senior Architect, Nokia Oyj, Helsinki. Keskustelu 17.4.2014.

Titanium Mobile Development Environment. 2014. Verkkodokumentti. Appcelerator Inc. <http://www.appcelerator.com/titanium/>. Luettu 17.4.2014.

Tizen An open source, standards-based software platform for multiple device categories. 2014. Verkkodokumentti. Tizen Project, a Linux Foundation Project. <https://www.tizen.org/>. Luettu 10.4.2014.

W3C. 2014. Verkkodokumentti. W3C. <http://www.w3.org/>. Luettu 14.3.2014.

Walker-Morgan, Dj. 2011. SproutCore 2.0 becomes Ember.js. Verkkodokumentti. <http://www.h-online.com/open/news/item/SproutCore-2-0-becomes-Ember-js-1394362.html>. Luettu 14.3.2014.

Ward, Robin. 2013. Co-founder, Civilized Discourse Construction Kit, Inc. <http://eviltrout.com/2013/03/23/ember-without-data.html>. Luettu 14.3.2014.

Why Vuforia. 2014. Verkkodokumentti. Qualcomm Vuforia. <https://www.vuforia.com/why>. Luettu 10.2.2014.

Wikitude About. 2014. Verkkodokumentti. Wikitude GmbH. <http://www.wikitude.com/about/>. Luettu 1.2.2014.

Wikitude Awards. 2014. Verkkodokumentti. Wikitude GmbH. <http://www.wikitude.com/about/awards/>. Luettu 1.2.2014.

Wikitude explained. 2011. Verkkodokumentti. Wikitude GmbH. <http://upload.wikimedia.org/wikipedia/commons/3/30/Wikitude_explained.png>. Luettu 13.4.2014.

Wikitude SDK API Reference. 2014. Verkkodokumentti. Wikitude GmbH. <http://www.wikitude.com/external/doc/documentation/latest/Reference/JavaScript%20Reference/index.html>. Luettu 17.4.2014.

Zuckenberg, Mark. 2014. Mark Zuckenberg. Verkkodokumentti. <https://www.facebook.com/zuck/posts/10101319050523971>. Luettu 10.2.2014.